Crazepony开源四轴飞行器

nieyong

Published
with GitBook

目錄

介紹 0
故事开始 1
- 我和Crazepony的那点事儿(1) 1.1
- 我和Crazepony的那点事儿(2) 1.2
- 我和Crazepony的那点事儿(3) 1.3
- 入手试飞指导 1.4
- 组装维修指导 1.5

四轴飞行器 2
- 四轴飞行器历史回顾 2.1
- 开源四轴飞行器 2.2
- 四轴飞行器明星产品 2.3
- 多旋翼 VS 直升机 VS 固定翼 2.4
- 大四轴 VS 小四轴 2.5

航模基本概念扫盲 3
- 航模常用术语 3.1
- 美国手 VS 日本手 3.2
- 三维中pitch，yaw，roll的区别 3.3
- 四轴飞行器的空气动力原理 3.4
- 无头模式 VS 有头模式 3.5

四轴飞行器基本组成部分 4
- 航模电机 4.1
- 电机驱动控制 4.2
- 航模桨叶 4.3
- 飞行控制器-主控MCU 4.4
- 飞行控制器——传感器 4.5
- 航模遥控器 4.6

Crazepony原理讲解 5
- Crazepony器件选型总览及说明 5.1
- Crazepony硬件原理讲解 5.2
- Crazepony软件框架讲解 5.3

Crazepony软件开发经验总结 5.4
二次开发指南 6
windows下开发环境搭建—裸机版本 6.1
windows下开发环境搭建—FreeRTOS版本 6.2
linux下开发环境搭建 6.3
Crazepony Android客户端 6.4
主控固件烧写 6.5
J-Link使用及常见问题 6.6
Crazepony上位机使用 6.7
自主悬停&高度融合 6.8
天线设计及遥控距离 6.9
摄像头和图传 6.10
Crazepony通信协议 6.11
飞行器平衡调试 6.12
动力效率和续航时间 6.13
四轴飞行器算法讲解 7
姿态解算简介 7.1
姿态的数学表示方法 7.2
姿态的测量 7.3
姿态解算流程 7.4
软件姿态解算 7.5
硬件姿态解算 7.6
四元数 7.7
三轴陀螺仪和三轴加速度计MPU6050 7.8
PID算法 7.9
气压计MS5611 7.10
博文精编 8
Carmack平方根函数经典案例 8.1
双曲函数和双曲线的故事 8.2
Crazepony鞭状天线室外飞行记 8.3
CrazePony与计算机视觉 8.4
Crazepony自平衡小车 8.5
DIY四轴飞行器如何入门？ 8.6
《大人的科学》引起的思考 8.7

Crazyflie研究笔记 9
FreeRTOS简介 9.1
介绍几个常用的宏的作用 9.2
固件系统流程框架 9.3
通信协议 9.4
其他 10
Crazepony的板形轮廓及沉金工艺 10.1
源代码的文件编码和文件格式 10.2
关于本书 10.3

Crazepony开源四轴飞行器

Crazepony项目旨在为大学生/航模爱好者/创客提供可二次开发的迷你四轴飞行器原型。我们秉承开放，合作，分享的理念，致力将Crazepony打造成航模爱好者学习交流的软硬件平台。Crazepony是一个完全开源的项目，包括源代码，原理图，设计思路等，并且提供详尽的百科知识库。可以通过它学习四轴飞行器相关知识，也可以在上面进行二次开发，实现自己的创意。

- 网址：http://crazepony.com
- Wiki：http://crazepony.com/wiki.html
- 博客：http://crazepony.com/blog.html
- Github站点：https://github.com/Crazepony

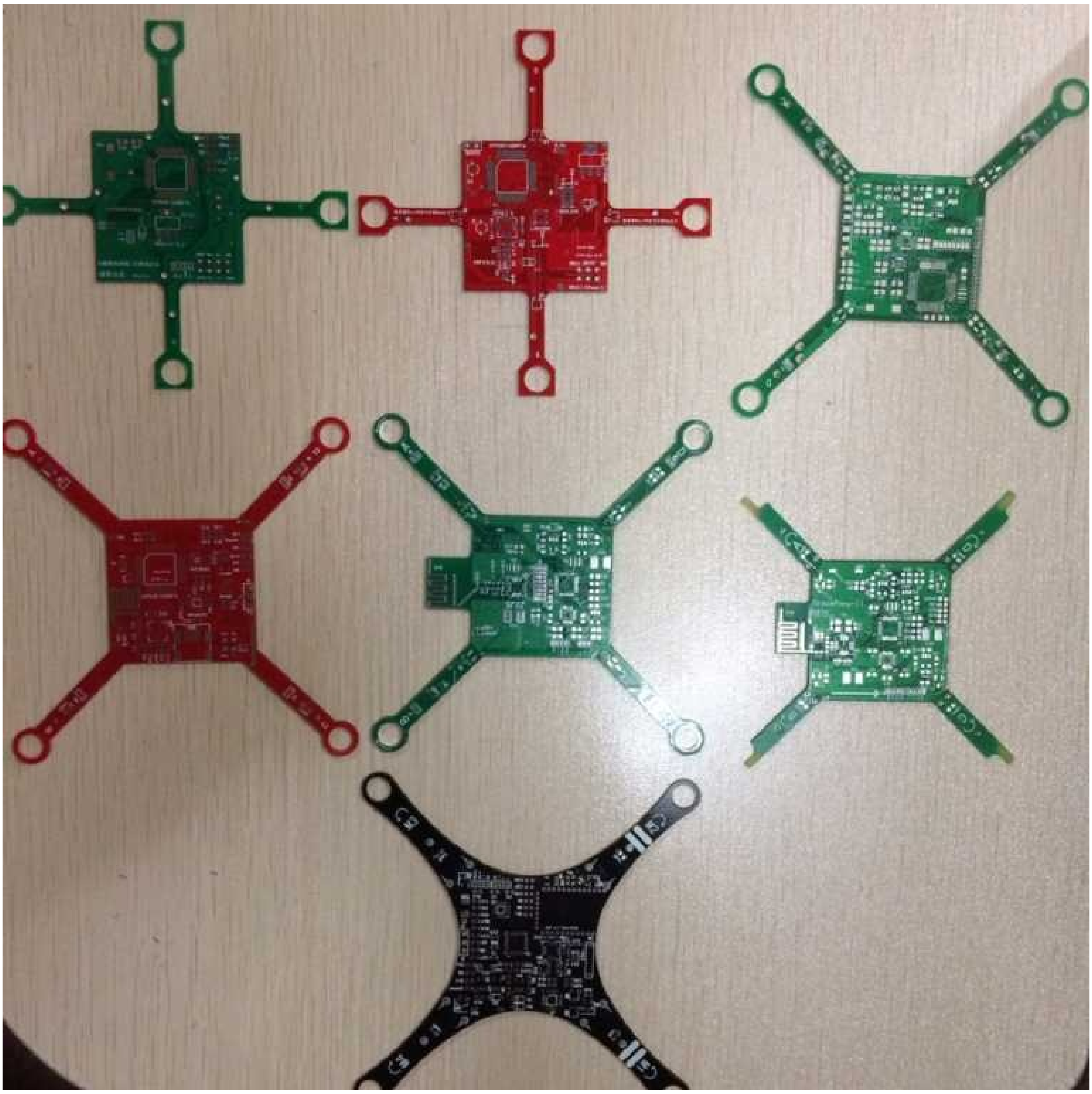


《Crazepony开源四轴飞行器》0.2版本。版本发布历史：

- 2015-2-7，第一次发布，0.1版本。

- 2015-11-24， 第二次发布，0.2版本。

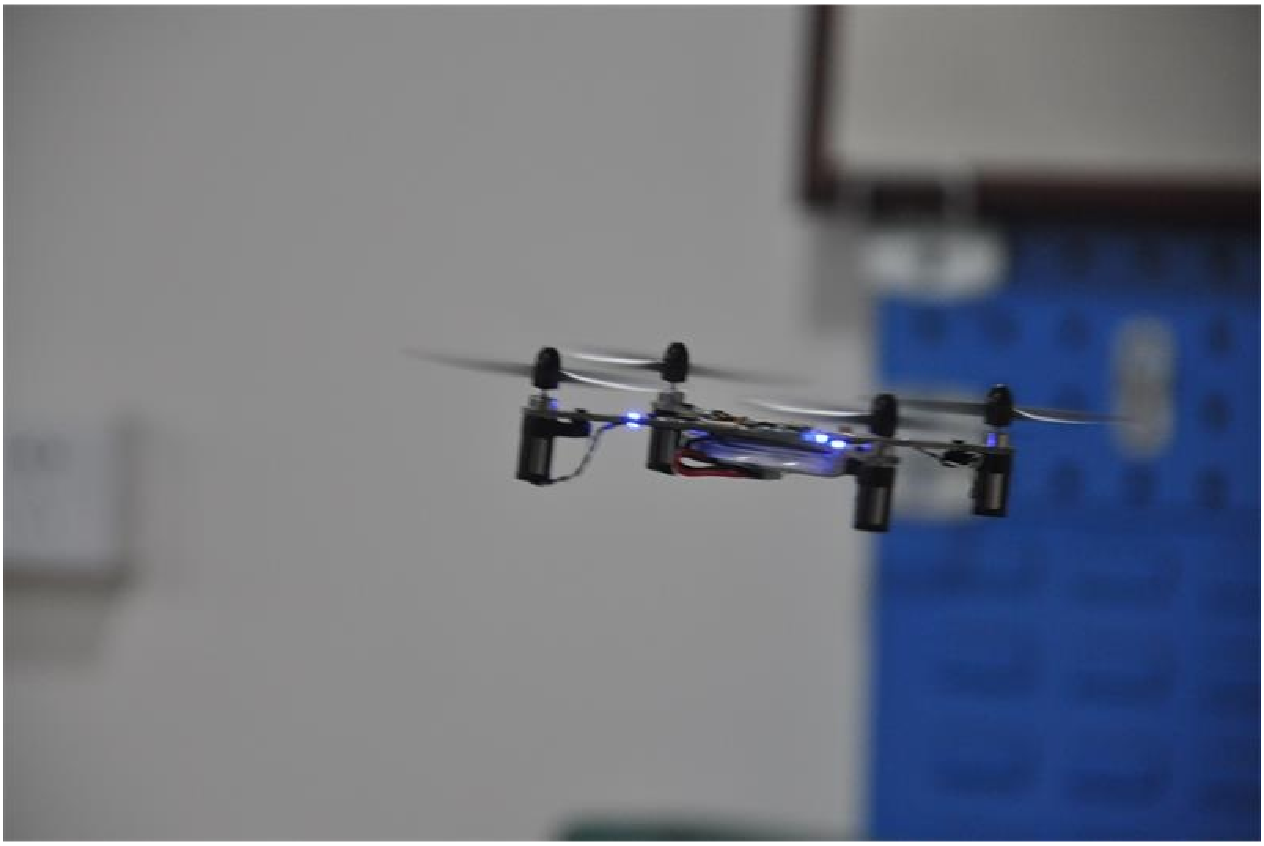
Crazepony项目起源于一个孩童从小对摆脱地心引力的梦想。在进入大学之后，终于能够把这个梦想和自己所学结合起来。到现在，Crazepony四轴飞行器已经在硬件上进行了10多次的改进，在软件，算法等方面经历了无数次的挫败和修改。



虽然它还有很多的不足，但是我们很自豪的将它推荐每一个朋友。因为它代表着我们对梦想默默的执着，对技术纯粹的追求，对挫折不言放弃的坚持。

Crazepony的理念

开放，合作，分享是Crazepony一直追求的理念，我们也把这种理念带入到产品的设计/研发/推广中。所以我们把Crazepony开源，希望能够为开源硬件/开源软件/创客领域贡献我们的一份力量。



我和Crazepony的那点事儿(1)

> 作者： CamelGo / March 5,2014

很荣幸，Crazepony的第一篇博客是我来写，鉴于是第一篇，按照惯例，我还是说点其他的先。

对现状的一些思考

在中国目前这个体制框架下，通过投机倒靶发家致富的人太多，而利用自己努力学到的知识和工程技巧解决问题的人屈指而数，少数人占有了大多数的社会财富，而他们往往又是那些肥头大耳，世俗圆滑的一群人。

当那些大型通信设备垄断商和房地产老总们通过资本运作和千军万马的工程密集型项目来聚敛巨额财富时，工程师的聪明才智和童年的幻想正一点点被压榨殆尽。创造力和想象力，不断屈服于自上而下的制度之中。而所谓的英雄，仅仅是这种企业的顶层，甚至他们都不一定懂得技术。



羡慕国外技术人员的纯粹，至少他们的初衷不是为了圈钱去做某个项目，去干某件事情，虽然最后都会因此而盈利，那是因为他们最开始的不计后果和奋不顾身，让他们后来得以发现这个产品的市场和价值。他们给我的感觉一直就是：“我没有其它目的，因为我爱它，所以我去做，赚不赚钱以后再说”也许这个观点带有个人主观色彩和片面的地方，但人家至少看起来是酱紫。

两次工业革命的起源都在西方，而在我们这片神奇的土地上，每天都在叫着创新 却每天都在滋生山寨，靠大量重复的廉价劳动力去复制，我们习惯了这种技术更新模式，都叫它借鉴，又好像总是要慢人家一个节拍。照这种趋势，毋庸置疑第三次工业革命的萌芽也在西方。

不思进取固然是可悲的，然而缺乏创新的土壤和原动力，同样没有希望。说这话的时候，其实自己心里也是满满的愧疚与不安，请接着往下看，心中默念“我只是借鉴而已…”。

一次关于微型四轴的演讲

2011年，宾夕法尼亚大学的VIJAYKUMAR教授在TED做了一次关于四轴飞行器的演讲，这次的飞行器只有手掌大小，其灵巧程度令人咋舌，演讲大部分时间在科普四轴飞行的原理，到最后教授展示了他们飞行器的应用，我惊呆了！它们可以编队飞行，多个飞行器合作搬运物体，合作弹奏一首音乐，进入建筑物内部飞行一圈，然后对建筑物内部空间结构建立三维模型并上位机重构等。这是让我内心难以平静的一段视频。



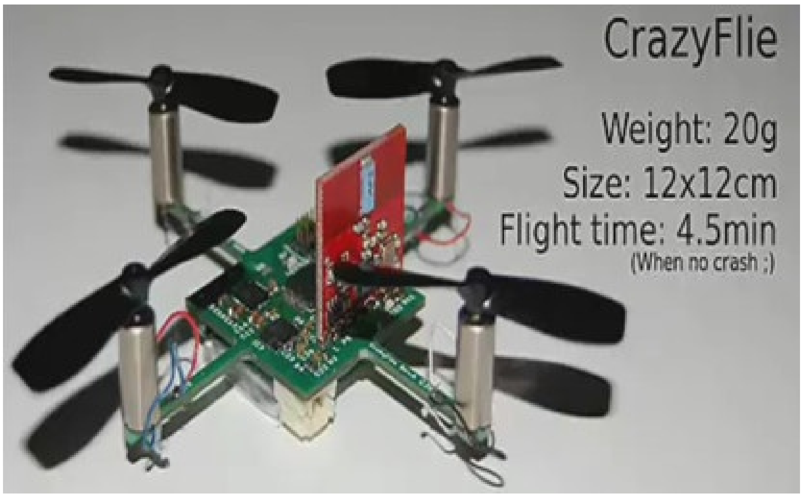
四轴编队协作搬运物体



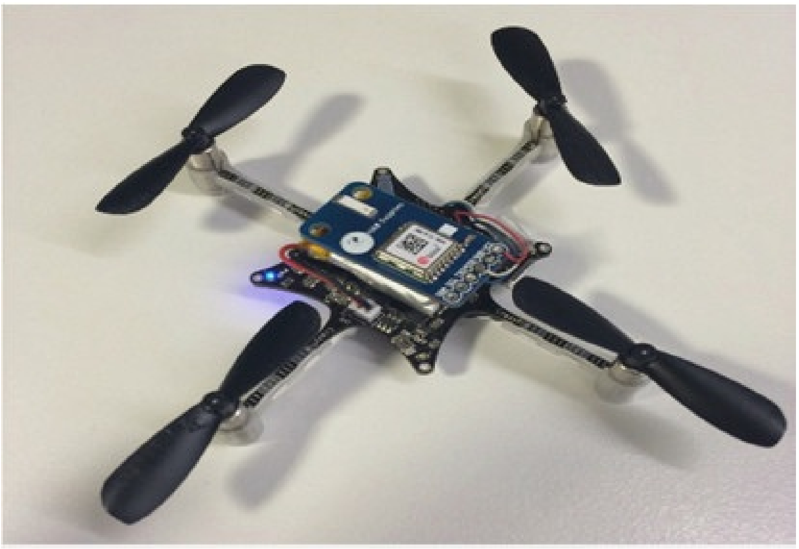
老美的技术已经发展到了令人发指的地步，很难想象这些技术要是被万恶的资本主义政府所利用，未来的战争将会是什么形式，战争的结局又将是怎样。

一个关于微型四轴的开源项目Crazyflie

偶然的机会，在果壳网看到了一个叫Crazyflie的微型四轴，这一次，我的心理防线被击得粉碎，因为它的开发者们看起来是跟我们年龄差不多的大学生，工作环境也是在一个类似工作室的地方，并不像宾大叫兽那么专业。他们第一版飞机是这样的，说实话，稳定性和外形设计不是那么好。



这是他们2014年最新版的飞机，较第一版好多了



这一次我被彻底点燃了，深陷其中，难以自拔，从那个时候开始谋划要做一个这样的小四轴，完成我童年那个对抗地心引力的小小梦想。

Crazepony四轴萌芽

就我个人而言，很喜欢小巧的电子产品,那种麻雀虽小,五脏俱全的感觉简直棒极了

要知道，把东西做大很容易，要把东西做小还要保证功能不减，这就需要耐心和体力了，仅仅是因为这样一个简单想法，促使了我去做这样一件看起来很无聊的事情。对这个飞行器的最初的构想是：机身一定是依托PCB板，大小一定不超过手掌，通信一定是我们熟悉的无线

解决方案，飞行体验一定要最好。同时，兼具娱乐性和开发性，让每个人都能在这里找到童年…

由于这是Crazepony的第一篇博客，写到这里就歇歇了，后续将会和大家交流一些开发过程中细节。由于笔者工程经验有限,前期调试过程中，走了不少弯路，很多问题考虑不够周全,还望能和各位一起探讨，共同进步。

```
2014年5月11日
```

我和Crazepony的那点事儿(2)

> 作者： CamelGo / May 29,2014

最近笔者忙于毕业事宜,博客没跟上...

上次说到机身硬件就停笔了，莫非是给以后留点故事来写罢了，好吧废话不多说了，那么就开始吧

（一）

首先谈谈硬件选型的依据吧：

主控方面：

曾经在犹豫用TI的430系列单片机还是意法半导的STM32。那是在我大二的时候，从来没有接触过STM32，以前都是用51单片机和TI的msp430系列单片机。好吧，我承认了，我做Crazepony其实是就是为了学习STM32的，没有买过STM32相关的开发板，就这么简单粗暴大刀阔斧的开始了我的STM32之旅...

最终选择用STM32当然还有其他原因，TI公司的MSP430系列都是基于低功耗在做文章，作为移动消费电子，对电源续航能力要求比较高的场合比较适用。

而意法半导的大部分单片机都是ARM架构体系下的Cortex内核为基础的单片机，这里有必要啰嗦一下何谓Cortex架构：

英国的ARM公司在经典处理器ARM11架构之后，为了给不同需求的CPU厂商提供服务，之后的内核架构命名都改为Cortex，并分成了A,R,M三类,也即将ARM的三个字母拆分为三个架构的名，代表着不同的发展方向：

- A系列处理器可托管丰富的OS平台和位应用商提供全方位的解决方案，诸如低成本手机、数字电视、机顶盒、打印机、服务器等
- R系列为实时处理器，要求可靠性、可用性、可维护性和实时响应的嵌入式系统提供解决方案
- M系列是一系列可向上兼容的高效能、易于使用的处理器，这些处理器旨在帮助开发者满足将来嵌入式的需要，这些需要包括低成本、不断增加的连接、代码改善移植等。M系列主要应用在智能测量、人机接口设备、汽车电子、工业控制、大型家电等。



很显然，苹果公司的A系列CPU也是嵌入的Cortex内核。

之所以选择STM32F103T8U6作为Crazepony的主控芯片。首先因为他是crotex-M3内核，继承了ARM的优良性能，主频能跑到72MHz，3个通用定时器，1个高级定时器，7通道DMA控制器，而且总线接口资源丰富；其次是因为它VFQFPN36的封装，只有6mm*6mm的占地面积,对这个寸土寸金的项目来说简直太赞了。这么高的性价比，当然让我选择了他作为主控。72MHz虽然生不了孩纸，但是足以处理除了图像之外的大部分任务了。

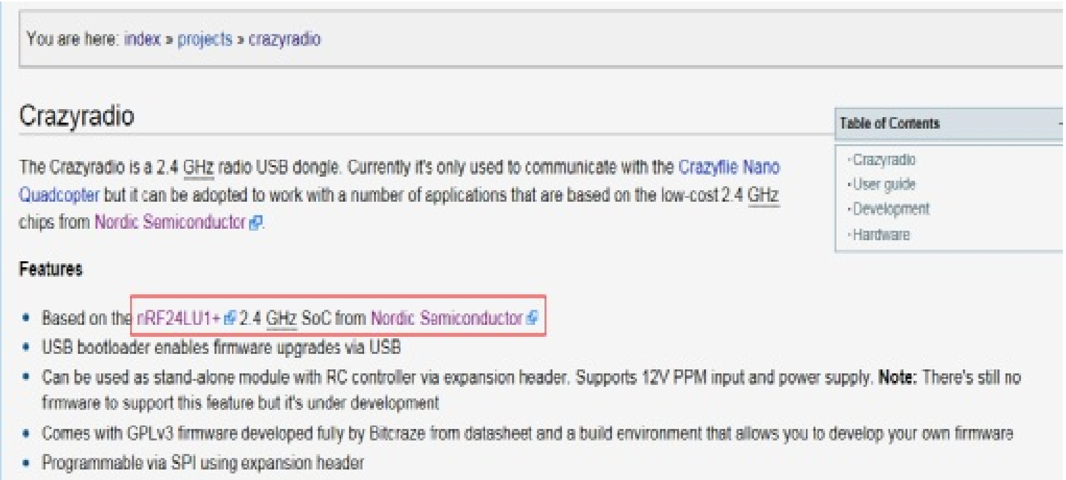
传感器方面：

目前Crazepony采用的是最常用的MPU6050陀螺仪加速度计一体芯片，成本不超过20元，对小四轴来说，它的精度和性能绰绰有余了（当我听说教研室师兄用的一颗传感器裸片卖1W+时，我整个人都不好了..），MPU6050在这个价位里面几乎是占有绝对的性价比优势。首先，它将陀螺仪和加速计整合在一个片上，通过IIC总线给出六个维度的ADC值；其次，芯片本身提供一个“从”IIC接口，供用户接第三方的IIC器件，一般选择是接一个电子罗盘，如HMC5883L,构成一个9轴的输出的姿态模组,现在MPU9150已经丧心病狂的把电子罗盘功能也

整合在片上了,但是要买60+元；最后，这颗芯片内部集成了一个DMP（Digital Motion Processor）处理器，这是最让我爱不释手夜不能寐的一个功能,直接硬件解算四元数,从某种程度上说解放了20%的主控资源

数据通信方面：

数据通信这块，也没什么特别要考虑的，短距离高速通信且免费这一点要求就限制了只能选择2.4GHz这个频段，在这个频段出了很多优秀的芯片厂商。在学生时代，我用得最多是Nordic公司的NRF24L01这个系列的收发一体芯片，由于刚开始着手启动Crazepony这个项目时，我只会这颗操作芯片，本着方便的原则，所以很自然的选择了这颗它（后来发现国外一个团队bitcraze也用的这个系列的芯片时，还是有点小激动的,不同的是，crazyfile用的是NRF24LU1，这颗芯片在与USB的接口上要容易些）



差不多了，硬件芯片选型大概是这样。下面谈谈这期间的一些细节

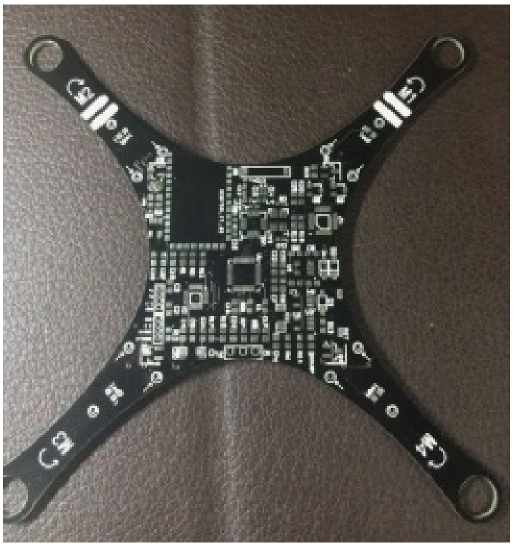
（二）

一个处女座仅仅是要求机身上芯片封装统一是QFN有错么,不要再黑处女座了好么，他们活得很累的。(~.~)

机身外形:

如上一篇博客谈到到的，我前前后后花了两年时间，以学习STM32这款单片为目的开始着手一个完全陌生的飞行器，从硬件到软件。可以想象，这两年我因为无知所走的弯路和画废掉的板子以及烧掉的生活费，加起来能换多少个菜包子和素馒头了。

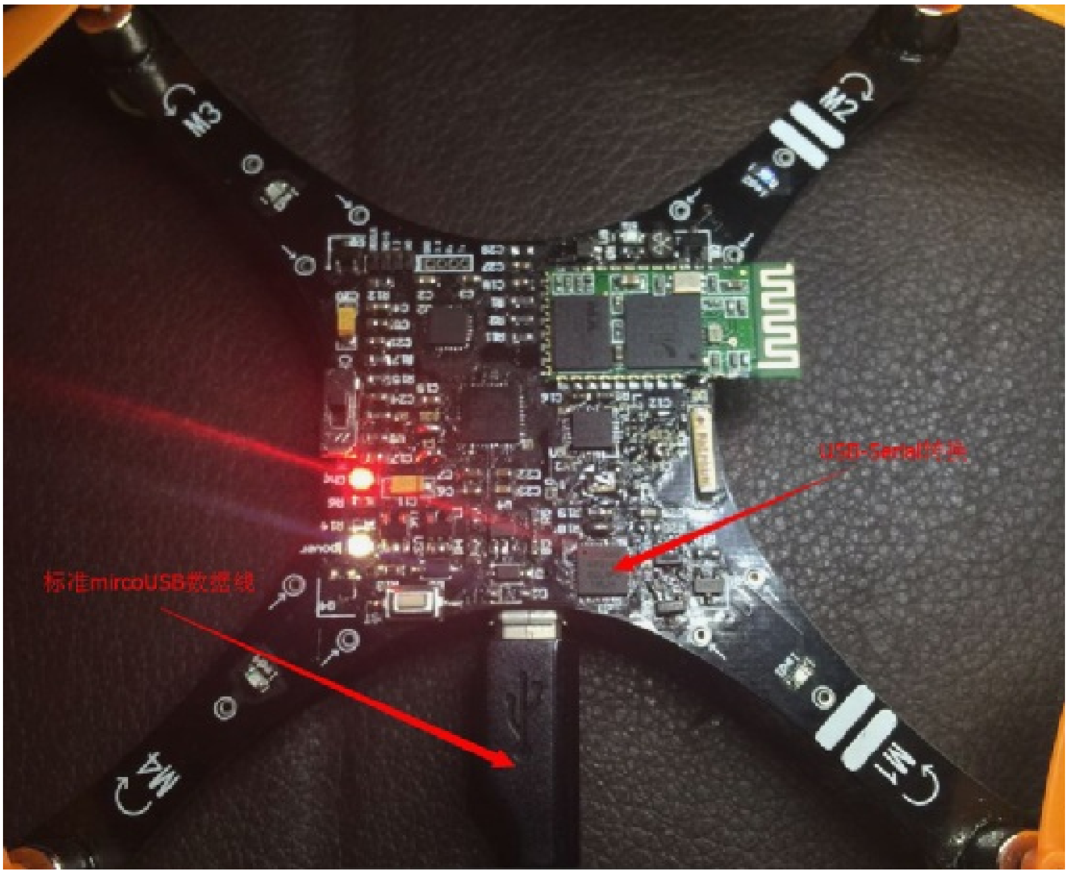
机身外形的设计对我来说是最头大的，一直想不到什么好的主意，最关键的是我本人是天线专业的，我自以为是的把2.4G的天线露在机身外部（如图1），凸显我是学过天线设计的男人。但是这严重破坏了美感不说，还强烈的刺激了处女座的用户。一次偶然的机会拆了一个平板电脑，发现主板上有根长得很像电容贴片电容的元件，百度后才发现，居然是一根2.4G的陶瓷天线，这….我这四年天线专业的所学算是全部还给老师了。于是，后来再参考了某公司的某飞行器外形，压缩了天线，有了最终双曲线过度的Crazepony外形（如图2）

调试方便的考虑:

在第三版之前,我都是用的stm32的SWD接口烧写调试固件代码,这种方式的确比较方便和有效,但是有个致命的弊端就是需要外接一个硕大的jlink调试器,这种设计简直太反人类了。纠结了好长时间，后来在看datasheet时发现STM32支持另外一种下载模式，用串口即可完成代码的下载，不足是不能在线调试。但是相比于能简化操作的诱惑，我还是决定在机身整合这样一个下载电路。（如图）

这样一来，机身和外部的有线接口就只有一根安卓手机的标配数据线mircoUSB线。它既是充电线，也是调参、烧写固件的数据线。这对大妈来说，想必操作也是很简单的吧



电机驱动:

由于笔者完全是由于一种强烈的爱好选择了飞行器,最开始连有刷电机和无刷电机的物理结构区别都不知道,电调又是啥?傻傻分不清楚…

从一个几乎零基础的状态去选择电机驱动芯片，弯路是必须要走的，学费是必须要交的。曾以为书上学到的东西马上就能用，马上能转化为产品，后来发现真的是自己想多了。

最开始用的三极管作为电机驱动，采用很经典的共射电路“三极管工作在开关状态应该就行了吧？”画了用三极管驱动的PCB板，发现电机越转越慢，根本没劲。“也许是因为三极管扛不了大电流，好吧那我换个中功率管吧，集电极最大6A电流行了吧？”可以想象结果是不行的

首先了解下为什么三极管作为简单的电机驱动是不可取的方案：

- 三极管作为一个古老的半导体先驱，它是以一个放大器件的姿态而出现的，它在线性区域特性集中，饱和与截止都是两种极端的工作状态，而作为电机驱动的话，我们只能选择它的这两种极端工作模式
- 用三极管作为大电流负载的驱动管时，不得不考虑的是他自身的管压降对负载的影响，这是很严重的。自身耗散越来越大，电机和管子是串联关系，电池电压只有3.7V，电机就只能越转越慢了

在晶体管家族里面还有一种跟三极管特性互补的，所有特性都集中在开关状态的晶体管，场效应管，即MOSFET。通常的场效应管完全导通时，源漏极电阻都是mΩ级别的，即它自身的耗散非常小。用它做为驱动管再合适不过了。

最终选择了一个SOT23封装的,导通电压Vgs<4v的场管（SI2302）,结果表现出了很好的驱动性能

```
未完待续~~ 2014/5/29
```

我和Crazepony的那点事儿(3)

> 作者： CamelGo / July 7,2014

毕业了,一直 沉浸在离别的情绪中难以自拔

距离上一篇博客更新,已经快有一个月了，如果再不写，那么Crazepony很可能被怀疑是烂尾了，所以...

今天主要谈谈以下几个事情：

- 软件的整体框架
- 第4版的修订版4.1版

关于软件

机身源代码部分，截止目前为止，都是属于一砖一瓦敲出来的裸机代码。为了满足各个层次用户的需求和体现出 我们的努力，后续会试着移植一个FreeRtos的实时操作系统内核，因为bitcraze 团队就用的这个内核，向他们无限靠近是我们的目标。

那么，现在就结合裸机代码，来说说Crazepony的软件框架：

学过 51单片机的都知道，任何一个处理器要正常运行后面的代码，首先必须得有一大段设备初始化的代码先运行，这些代码用于初始化处理器的内部时钟、中断优先级、I/O 口的输入输出方向等等，也就是为后续代码正常运行，做了一个环境配置准备。

Crazepony 的主控是 Crotex-M3内核，其实就是ARM架构发展到一定阶段的产物，那是什么呢？还是ARM架构。于是，对ARM的初始化，首先必须要做的就是系统时钟初始化，中断向量表初始化，中断优先级初始化，I/O方向初始化，如下：

```
/**********************************************
                飞控主函数入口
功能:
1.初始化各个硬件
2.初始化系统参数
3.开定时器4等待数据中断到来
4.开定时器3串口广播实时姿态以及相关信息
**********************************************/
int main(void)
{

  //SystemClock_HSE(7);            //系统时钟初始化，时钟源外部晶振HSE
  SystemClock_HSI(9);            //系统时钟初始化，时钟源内部HSI
  UART1_init(SysClock,115200);   //串口1初始化
  NVIC_INIT();                   //中断初始化
  STMFLASH_Unlock();             //内部flash解锁
  LedInit();                     //IO初始化
  delay_init(SysClock);          //滴答延时初始化
  BT_PowerInit();                //蓝牙电源初始化完成，默认关闭
  MotorInit();                   //马达初始化
  BatteryCheckInit();            //电池电压监测初始化
  IIC_Init();                    //IIC初始化
  MPU6050_DMP_Initialize();      //初始化DMP引擎
  PID_INIT();                    //PID参数初始化
  ParameterRead();               //Flash参数读取
  NRF24L01_INIT();               //NRF24L01初始化
  SetRX_Mode();                  //设无线模块为接收模式
  //PowerOn();                     //开机等待
  BT_on();                       //蓝牙开
  TIM3_Init(SysClock,50);        //定时器3初始化，调试串口输出
  TIM4_Init(SysClock,1000);      //定时器4初始化，定时采样传感器数据，更新PID输出
  while (1);                      //等待数据更新中断到来

}
```

然后，STM32内部模拟EEPROM初始化；然后，LED初始化；然后，延时函数初始化；然后，蓝牙电源使能初始化；然后，电机PWM输出初始化；然后，电池电压AD初始化；IIC总线初始化，传感器初始化，PID参数初始化，无线收发模块初始化为接受模式，开 蓝牙，开定时器3，开定时器4。

初始化看起来很繁杂，很多，也没啥好说的。现在谈谈Crazepony是怎么正常飞起来的：

前面已经努力的初始化了一大堆东西了，那么飞机起飞所需设备以及环境也正常了，Crazepony接下来需要做的最重要事情，就是如何去合理的控制这些设备，让飞机正常起飞

飞机起飞所需要的必须设备就是：电机+螺旋桨+姿态传感器。

在这里，我们必须承认的一点就是，四路PWM信号的占空比大小，线性的代表了电机转速快慢，也即线性代表了电机上提供的升力。于是，Crazepony现在只需要控制单片机内部定时器准确的输出四路PWM信号就行了，其他的事情，交给执行机构：电机+螺旋桨去完成。



接下来，程序运行到死循环while（1）；程序会一直停在这里,等待数据中断的到来，而不是死机死在这里，这是有区别的，学过51的人都知道，我不在多说。

在初始化代码段，我们说到初始化了两个定时器，一个定时器3，一个定时器4，这两个定时器都可以打断死循环while（1）。定时器3用于广播机身姿态信息，定时器4的任务要繁重得多，用于 更新遥控数据+机身姿态融合+PID计算输出+PWM输出。（关于PID和姿态融合部分的细节http://www.crazepony.com/wiki.html ）可以看到，定时器4里面任务的优先级明显要比定时器3实时性要求更高，所以。中断优先级的顺序是：定时器4 > 串口中断 > 定时器3。姿态更新频率为1000Hz，广播信息更新频率为1Hz。

1000Hz的数据更新频率,已经足够承受Crazepony反应了,阿莫上有人说几百赫兹是最佳。其实，这差不多了多少。学信号的人都知道，采样间隔越小，对信号的还原越精确，所以，我还是宁愿高一点



综上，有点乱，但是我们缕一缕。很简单，只有3个中断。定时器4是核心中断，所有的算法都是在这里实现的，机身的稳定也是靠这个中断来实现的

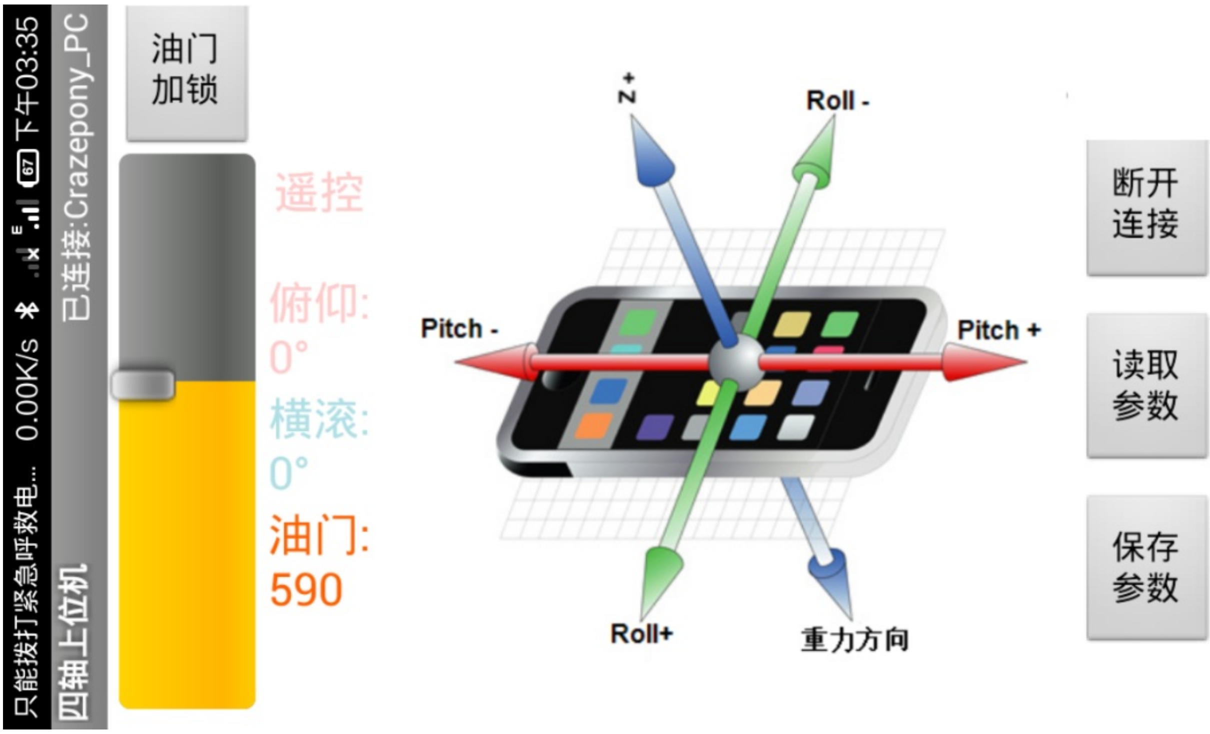
大概的框架就是这样,实现的细节，源码里面都有注释，写得很浅显易懂

关于第四版的修订

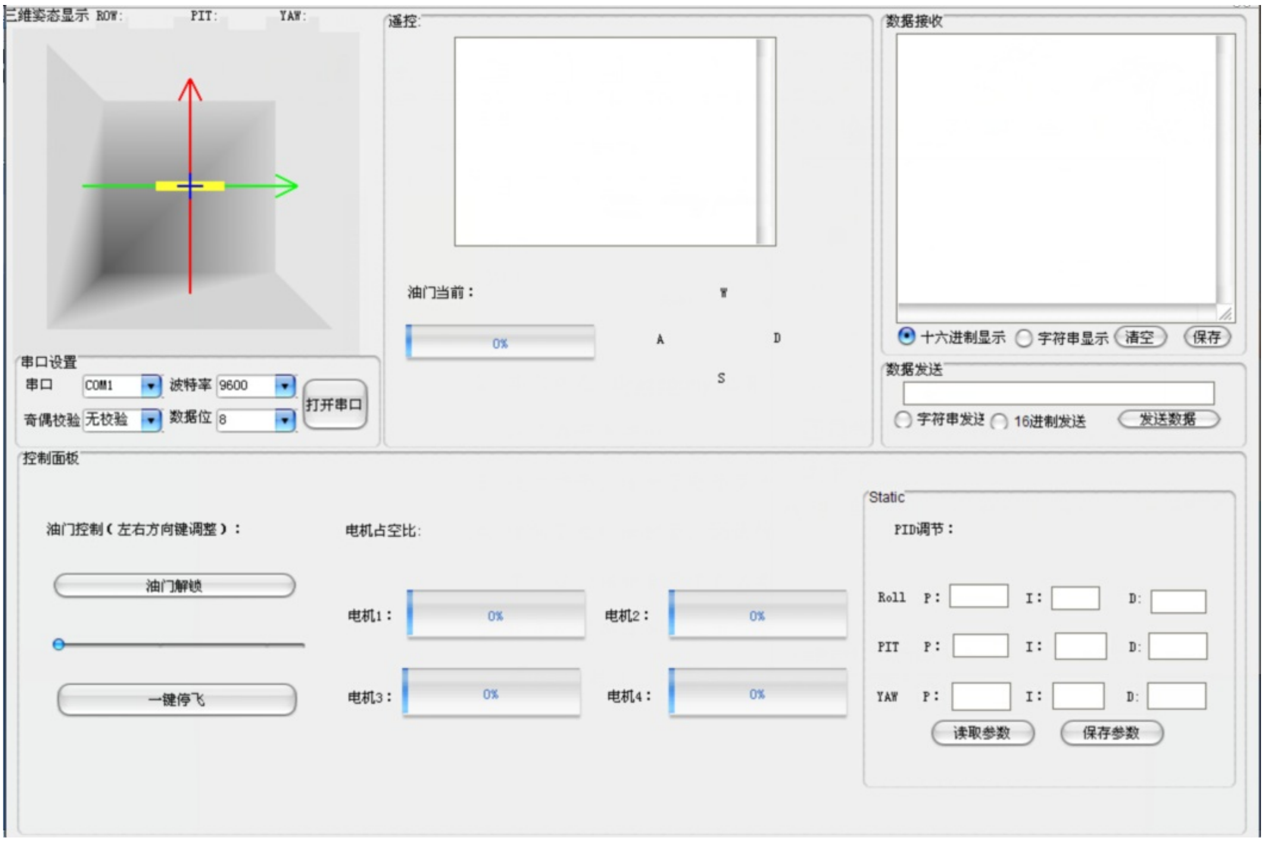
按照原计划，第四版将会是最终版，不会再改了。但是鉴于嘉利创工艺的问题，我还是得做4.1版。这里不吐槽嘉利创了，大家都不容易~~

说说4.1版的特性吧：

- 最明显的一个特性就是省略了实体遥控，改用安卓app客户端来控制Crazepony，直接用手机的姿态去控制Crazepony的姿态，屏幕上只有 一个油门键，没有方向键，界面如下：



- 其次就是，Crazepony也有了类似地面站的东西，目前正在开发中…



- 硬件方面，增加了电子罗盘HMC5883L
- 增加了电机保护套，调试阶段，电机屁股经常被戳穿，这个保护套是很有必要的
- 机身所有焊盘全部采用沉金工艺，阻容元件全部采用0402封装，且全部工厂贴片加工，布局合理



最后，有必要说下，4.1版，我们目前正在加班熬夜完善，争取在9月初公布购买链接

Crazepony四轴交流群 ：346226561

```
~~~~未完待续 2014/7/7
```

入手试飞教程

> 作者：nieyong 本教程适合Crazepony 5.2，5.1版本

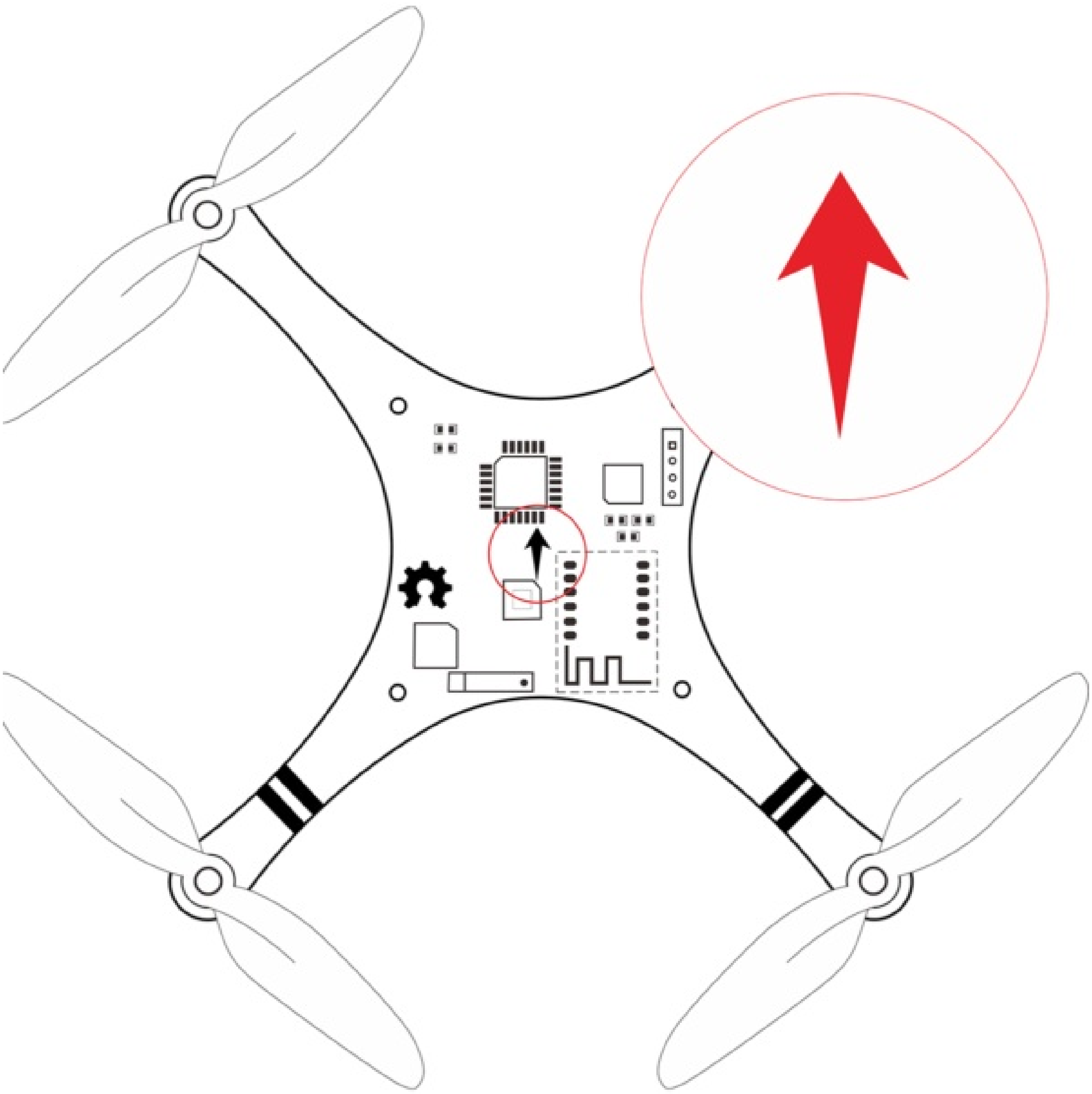
整机物料列表

- Crazepony四轴飞行器，电机/桨叶已经安装（1个）
- Crazepony遥控器，摇杆/摇杆帽已经安装（1个）
- 备用电机2个

确认飞机的前与后

Crazepony默认固件使用是的X结构模式，无头飞行模式，遥控默认固件为左手油门（美国手）。

将飞机水平放于地面，在飞机正中间，有箭头指向飞机的正前方（飞机的开关方向）。除此之外，还可以通过机臂确认前后。M3和M4机臂为飞机机头方向，M1和M2为飞机机尾方向。



一般我们站在飞机的后方，面向飞机开始操控。

> Crazepony遥控默认固件为美国手，无头模式。更多关于遥控器美国手。更多关于无头模式飞行。

遥控器操作

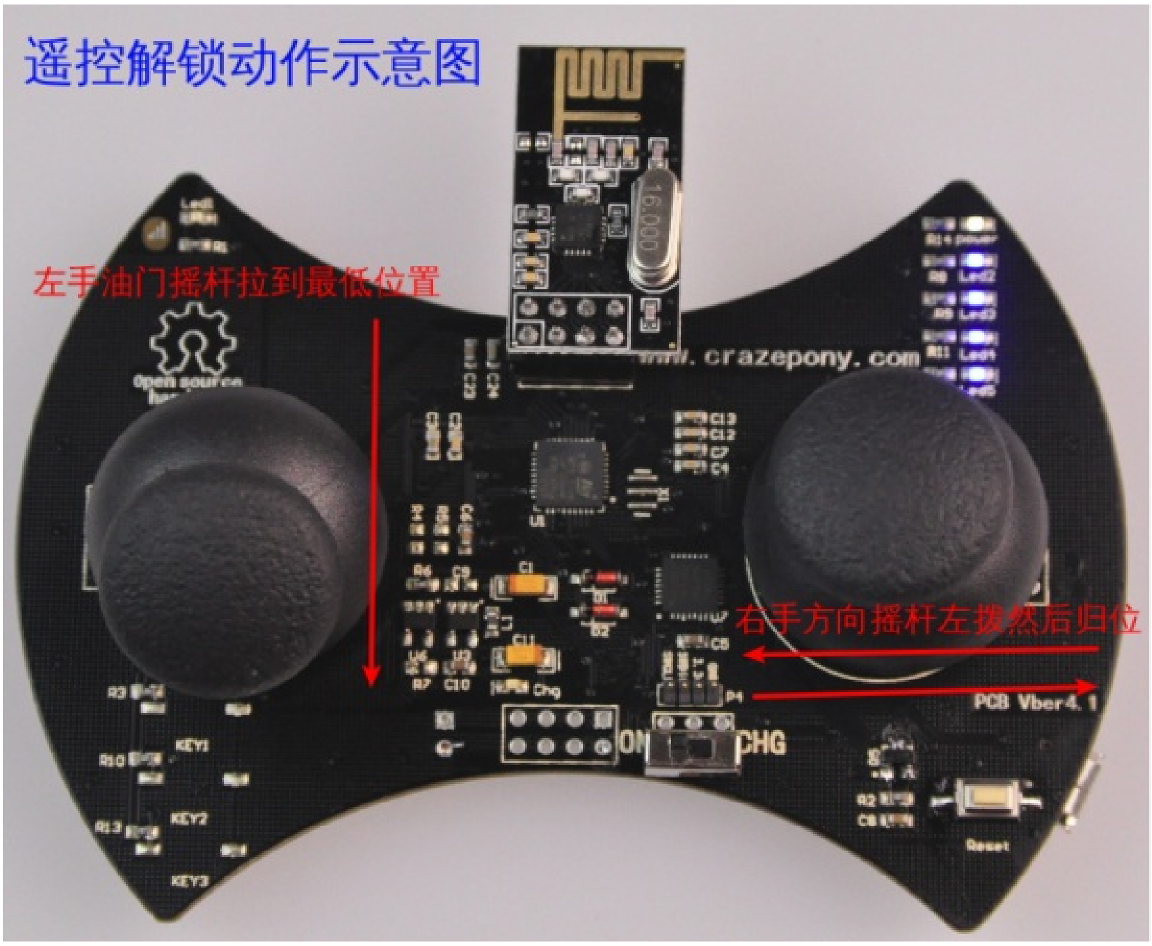
- 将遥控器开关推到ON。
- 将飞机的开关推到ON，飞控开始初始化，大概需要10秒左右。最后如果M1和M2机臂两个LED闪烁，则可以进入下面的解锁试飞。否则，请进入异常处理部分，具体查看下面的Crazepony灯语部分。
- 解锁试飞。左手油门低到最低同时右手横滚拉到最左边，解锁油门。解锁成功，飞机电机将怠速旋转。（最新版本遥控固件不再需要该步骤）
- 按下遥控器上“-”按键，M1和M2机臂上LED常亮进入校准。校准结束，M1和M2机臂上

LED开始闪烁。按下遥控器上的"+"按键，开启电机怠速转动。

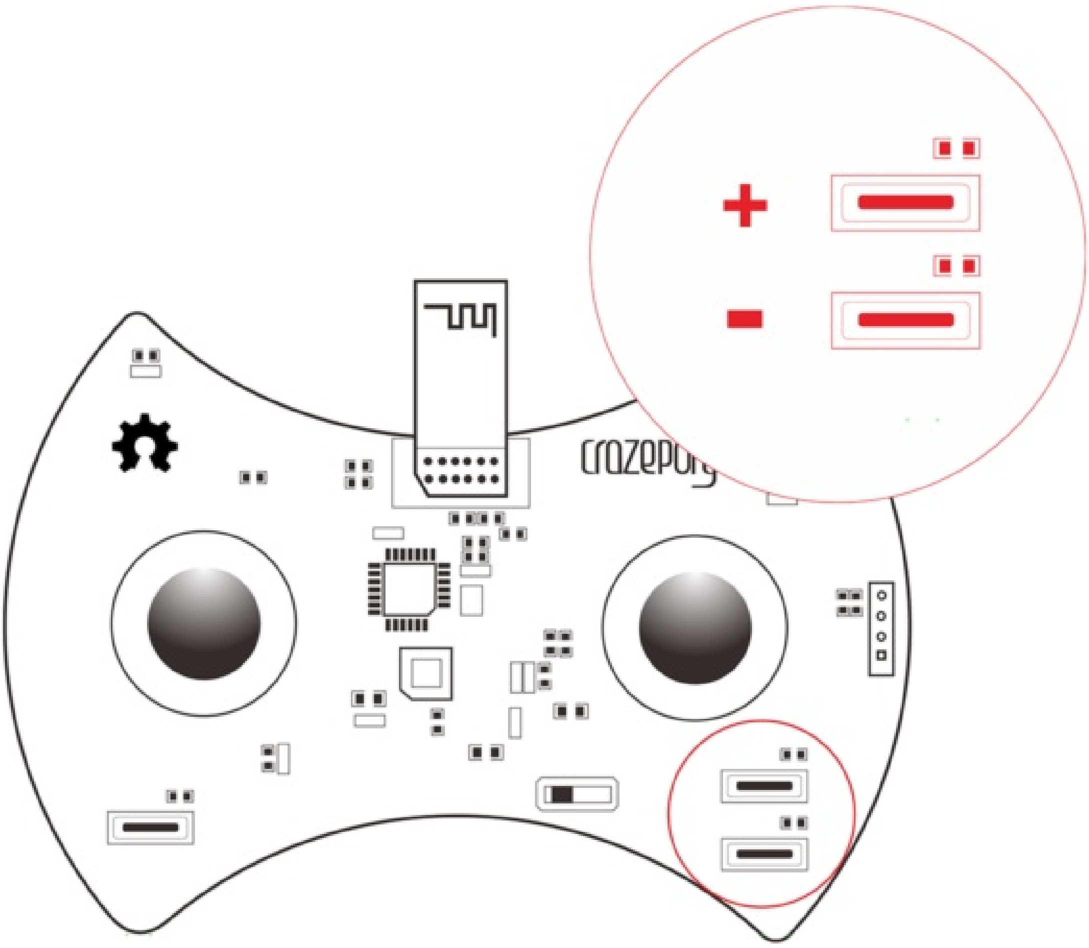
- 默认是定高飞行模式，油门通道控制垂直方向的速度，缓慢推油门，四轴将开始上升，拉到油门中位以下则下降。
- 降落操作，可以使四轴下降至地面，然后，按"+"号键关闭电机。

> 默认飞行模式为无头、定高飞行模式。飞机记录起飞的方向，飞行过程中，飞机响应摇杆的方向，始终以起飞时的方向为准，没有机头机尾的区别。

下面为遥控器解锁操作示意图。



下面为遥控器上按键功能标注。



手机APP遥控-蓝牙4.0 BLE

Android手机需要**Android 4.3**版本以上。

官网下载支持BLE的Android APP，并且安装。另外，因蓝牙版本的设计及本批次选用扭力稍大于之前版本的720空心杯电机，我们修改了悬停两处参数，最新版适用于蓝牙4.0机身、遥控固件下载链接。

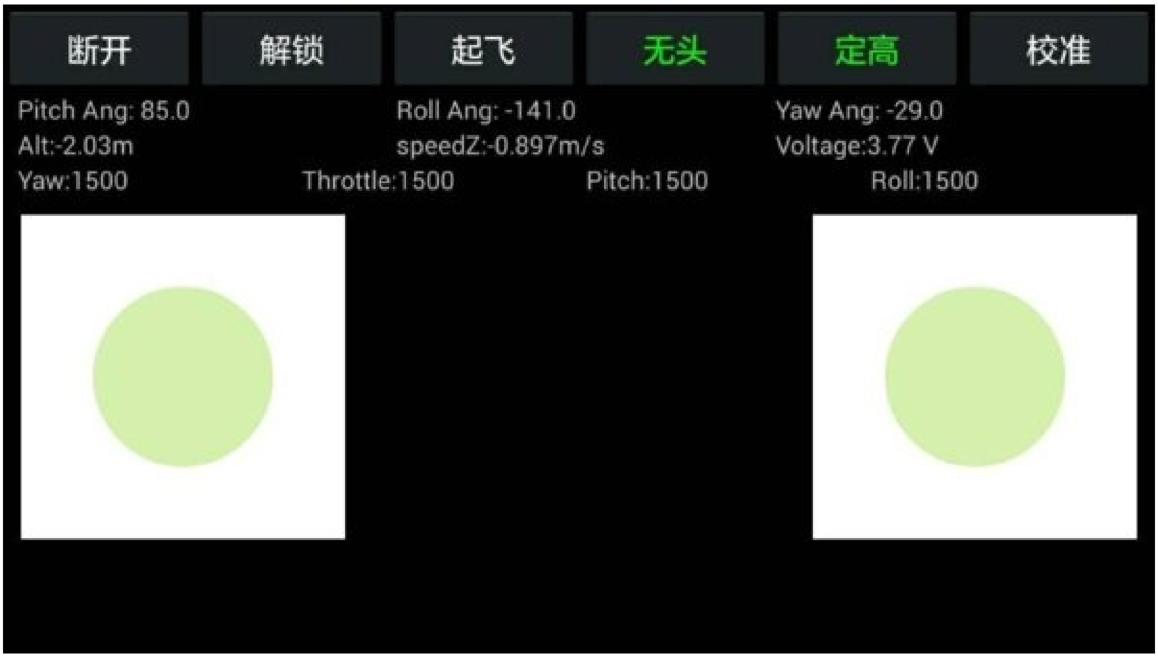
蓝牙4.0版本APP使用方法

- 将飞机的开关推到ON，LED开始进入一系列的闪烁变化，飞控开始初始化。
- 打开APP，点击连接。
- 查找设备，会搜寻到名字为Crazepony设备。点击连接此设备，无需配对，无需**PIN**码，连接成功后，机身的蓝牙提示灯常亮。

- 点击校准按钮，此时M1和M2臂上的led灯会常亮，校准陀螺仪和加速度计，校准完成后开始闪烁，等待解锁。
- 点击定高按钮。
- 点击解锁按钮，四个电机怠速旋转，推动油门，飞机起飞，松开油门，飞机悬停。
- 点击起飞/降落按键，可以手动降落飞机（起飞按键目前不支持此功能，降落按键有功能）。

注意：用app操作时，飞行模式需要手动设置，即需要点定高和无头才能实现相应的功能。

遥控和app控制，都支持失控自动降落功能，控制距离建议不要大于20米，最好在室内练习熟练以后再去室外飞。





iPhone手机APP很快就可以发布。

手机APP遥控-蓝牙2.1

蓝牙**2.1**版本现在已经停产，现在支持蓝牙**4.0 BLE**。

官网下载支持蓝牙2.1版本的APP，并且安装。

- 将飞机的开关推到ON，LED开始进入一系列的闪烁变化，飞控开始初始化。
- 打开app，点击连接。
- 查找设备，第一次连接crazepony，设备名是一个叫null的设备，点击连接此设备，会显示出完整设备名：crazepony，PIN码为：**1234**，连接成功后，机身的蓝牙提示灯常亮。
- 点击校准按钮，此时M1和M2臂上的led灯会常亮，校准陀螺仪和加速度计，校准完成后开始闪烁，等待解锁。
- 点击定高按钮。
- 点击解锁按钮，四个电机怠速旋转，推动油门，飞机起飞，松开油门，飞机悬停。
- 点击起飞/降落按键，可以手动降落飞机（起飞按键目前不支持此功能，降落按键有功能）。

注意：用app操作时，飞行模式需要手动设置，即需要点定高和无头才能实现相应的功能。

遥控和app控制，都支持失控自动降落功能，控制距离建议不要大于20米，最好在室内练习熟练以后再去室外飞。

Crazepony灯语

Crazepony的4个臂膀各有一个蓝色的LED。我们规定了部分LED闪烁的方式（灯语），用于指示飞机的状态。灯语分为指示类型和异常类型，指示类型用于开发和调试，异常类型则必须要处理才能够使用飞机。

异常指示灯列表。

| 灯语|含义|解决办法| |旋转闪烁|未接收到遥控信号（遥控或者手机APP）| 开启遥控器。如果开启遥控器未能解决，则需要先开启遥控器，再开启飞机重试（重烧固件时会遇到）| |M1 M2和M3 M4交替闪烁|IMU未校准|将飞机水平放置地面上，按遥控器上“-”号键（或者APP上的校准按钮）进行IMU校准| |4灯同时快闪|低电压提示|给电池充电|

正常提示灯列表。

| 灯语|含义| |M1和M2常亮3秒，M3和M4长灭|IMU启动校准，校准陀螺仪| |M1和M2闪烁，M3和M4长灭|正常，可以进行起飞，IMU校准等动作| |M1闪烁8次，其它灯长灭|配置蓝牙模块|

> 关于蓝牙配置：机身的M1臂上的LED灯会闪烁8次，这里是在检测蓝牙模块是否正常；如果蓝牙设备波特率不是设定值（115200），那么会进入写蓝牙设备名，写波特率，写PIN码的逻辑，M1,M3和M2，M4臂上的灯会交替闪烁，配置蓝牙参数成功后，四个灯一起点亮1秒

电量检测和充电

飞机的四个机臂LED同时出现快速闪烁，则表示飞机电池需要充电。将飞机开关拨到Charge位置，连上usb线则可以充电。可以使用电脑，充电宝，手机充电器等为飞机进行充电。



充电时红色LED亮起，变暗或者灭掉的时候则表示充电完成。

电机容易损坏的问题

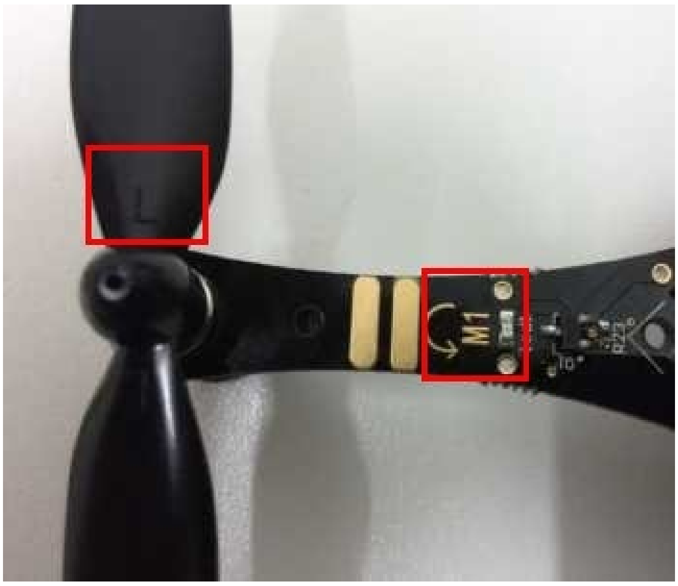
为什么Crazepony的电机臂部老是被戳穿？有人会在这个时候开始一个劲儿乱喷骂娘，我也很无奈。

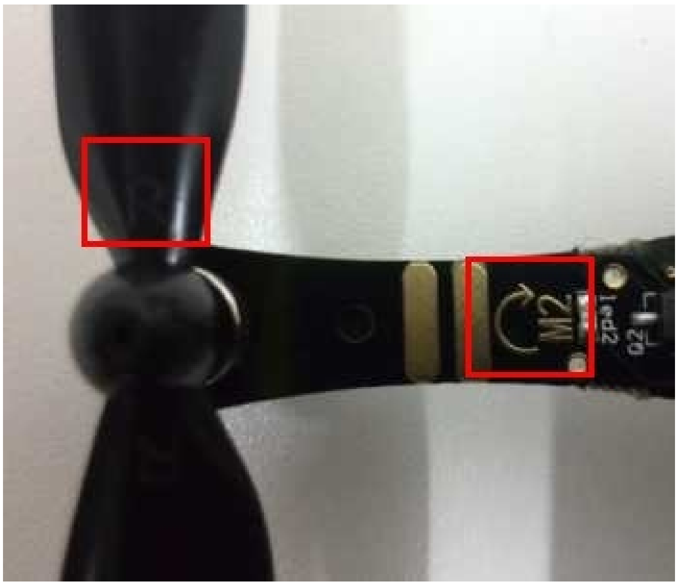
这么说吧，正常情况下100次起落，都不会对飞机造成任何物理结构上不可逆的毁坏。那么，电机臂部被戳穿这种事情，属于飞机脸先着地的意外，他的臂部是被那根轴顶穿的，空心杯电机其实很脆弱，这个我没办法控制，我已经尽力了。我看了国内的那些用空心杯电机的小四轴，好像都有这个问题...所以，大家提高飞行技术，是对Crazepony最直接的保护。当然，电机坏了，这种事情太好处理了，多准备些空心杯电机，只要主板没坏，随便怎么折腾都可以...

桨叶/遥控安装

> 对于Crazepony 5.2版本，入手即飞，不需要该步骤的安装。

四轴飞行器的桨叶分为两种，正桨和反桨。安装的时候不能够搞混淆。桨叶上标示有R的安装到M2，M4两个机臂的电机上。标示有L桨叶安装到M1，M3两个机臂电机上。





发货清单中有黑色和透明两种硅胶垫，用于缓冲飞机降落时的撞击，并且起到保护电机线的作用，请自行选择一种安装。效果如下图所示。



你拿到Crazepony的包裹的时候，需要自行安装遥控器摇杆帽，没有左右之分。

将2.4G模块安装到遥控器上的排针插座上。模块方向向外。

散件组装教程

> 作者：nieyong 本教程适合Crazepony 5.2，5.1版本

从5.1版本开始，Crazepony不再提供散件销售。本组装教程主要目的是提供给后期的维修和改装。

物料列表

- Crazepony主控电路板（PCBA）
- 720空心杯有刷电机
- 75mm正反桨叶
- 有刷电机套
- Crazepony遥控器主控板（PCBA）
- 380mAh航模电池
- 摇杆帽
- 遥控器用2.4G无线模块（1个）
- 蓝牙透传模块HM-06（1个）

> PCBA是英文Printed Circuit Board（PCB）+ Assembly的简称，也就是说PCB空板经过SMT贴片上件，再经过DIP插件，最后得到的元器件已经贴好的电路板，简称PCBA。

需要电烙铁，螺丝刀，剪刀（或者小刀）等工具，需要有一定的DIY能力。

电子软件部分测试

Crazepony主控电路板/遥控器电路板已经烧入了最新版本的代码，并且进行了测试。用户拿到之后可以通过以下方式对其基本功能进行测试，确保正确。

todo

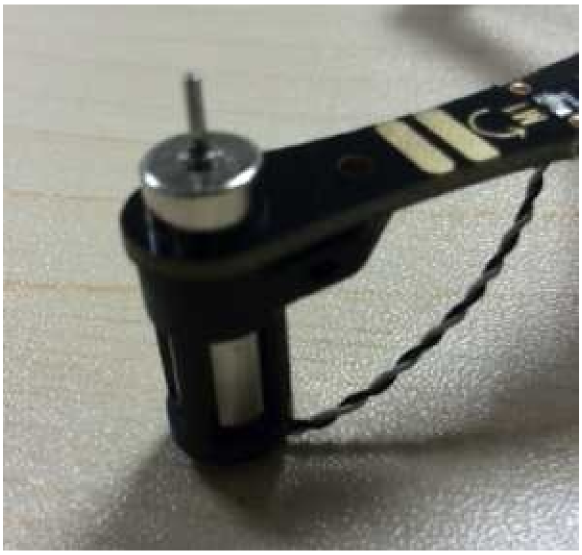
电机安装

将电机插入电机套内，电机供电线从电机套底部留出。注意电机一定要完全插入底部，不要留有空隙。否者，电机套保护电机底部免受挫穿的作用就没有了。



将电机供电线扭转成麻花状，留足到电机焊盘长度，剪去多余部分，最后将电机线焊接到焊盘上。电机的焊接，对应原则是，将黑线焊接到标示有+的焊盘，白线焊接到另外一个焊盘。

将电机露出电机套部分插入飞控电路板的电机孔中。这次的电机孔刚好能够卡住整个电机。



在发货清单中提供有2M螺丝和垫片，用于固定电机和电机套。





将电机套底部开一个缺口，将电机线从这个缺口引出。然后粘上提供的黑色（或者透明）的硅胶垫。这样在多次起降撞击的时候，不会伤到电机供电线。



到现在，整个电机的安装就完成了。

电池的安装

将电池焊接到飞控板上，注意电线留出合适的长度，一般是固定在飞控板的下面，所以线是焊接到下面的。电池的红色接飞控板的3.7v焊盘，黑色接GND焊盘。在电池上，我们已经沾上了双面胶，将胶纸的保护膜撕掉，将电池固定于飞控板正中央。



最近发现有小伙伴尽然将电池放到了PCB的上面，而且用绳子固定的（其实上面已经沾了双面胶）。这让我久久不能平静，确实是我们文档说得不够详细。PCB的上面有基本上所有的元器件，容易造成短路，而且可能遮盖了天线，导致遥控的距离缩短。下面就是几个详细的

电池固定示意图。最好是固定到**PCB**的下面。

将双面胶撕开，准备粘贴。



将电池按照如图所示的方位粘贴到PCB的底侧，并且压紧。



最后效果图。



下面部分，参考《入手试飞教程》。

四轴飞行器

四轴飞行器是航模无人机中最常见的一种，为众多的航模爱好者所喜欢。在很多科幻电影中也常有四轴飞行器的出现。美国电影《复仇者联盟（Marvel's The Avengers）》中的空中航空母舰就是一艘巨大的四轴飞行器。



四轴飞行器历史回顾

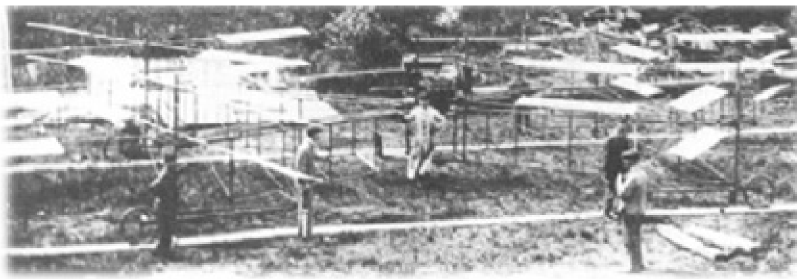
> 作者：nieyong

四轴飞行器是微型飞行器的其中一种，也是一种智能机器人。

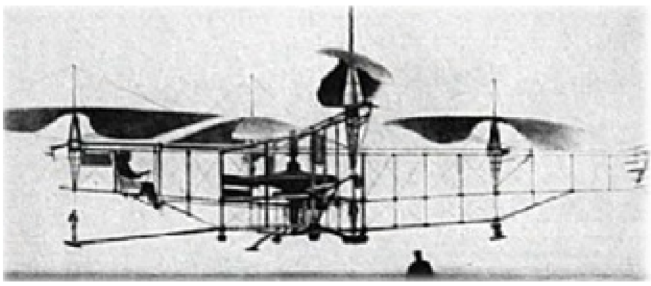
1907 年，法国Breguet兄弟制造了第一架四旋翼式直升机，这次飞行中没有用到任何旋翼式直升机，这次飞行中没有用到任何的控制，所以飞行稳定性是很差。



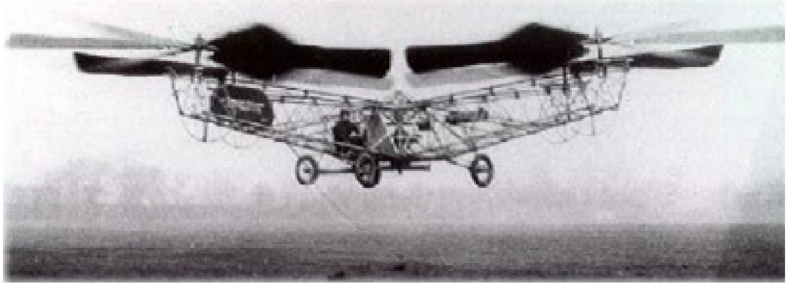
1921年，George De Bothezat在美国俄亥俄州西南部城市代顿的美国空军部建造了另架大型的四旋翼直升机先后进行了一架大型的四旋翼直升机，先后进行了100多次的飞行试验但是仍然无法很好的控制其飞行，并且没有达到美国空军标准。



1924 年，出现了一种叫做Oemichen的四旋翼直升机，直升机首次实现了1km 的垂直飞行。



1956 年，Convertawing造了一架四旋翼直升机，该飞行器的螺旋桨在直径上超过了19 英尺，用到了两个发动机，并且通过改变每个螺旋桨提供的推力了来控制飞行器。



美国陆军研制的VZ-7，被称做Flying Jeep，有效载荷250KG，425马力涡轮发动机驱动，容易起飞，但不能满足军方速度和高度要求，1960年被退还给发明人，美国航空事业先驱-柯蒂斯。维基百科-Curtiss-Wright VZ-7。



在此之后的数十年中，四旋翼垂直起降机没有什么大的进展。近十几年来，随着微系统、传感器以及控制理论等技术的发展四旋翼垂直起降机制理论等技术的发展，四旋翼垂直起降机又引起人们极大的兴趣。研究集中在小型或微型四旋翼飞行器的结构、飞行控制以及能源动力等方面。

开源四轴飞行器

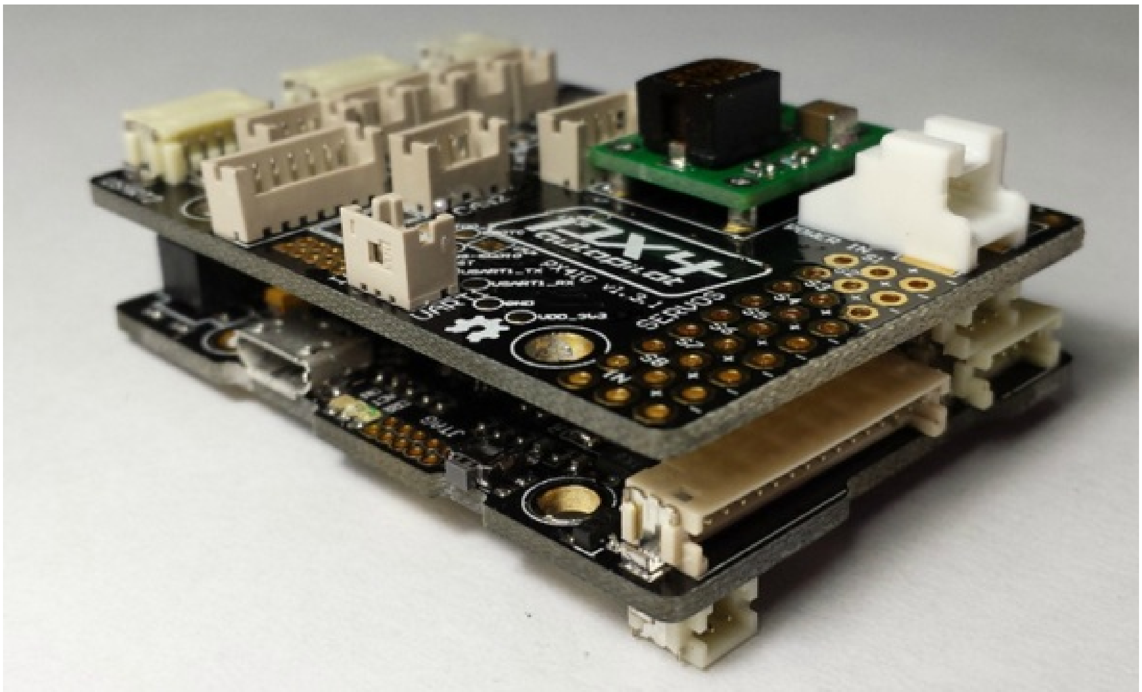
> 作者：nieyong

APM & 3D Robotics

3D Robotics公司旗下的飞控有3款，分别是ArduPilot（简称APM），PX4和Pihawk。

APM是比较古老的版本，处理核心使用的是Arduino，16位mega系列单片机，开发环境为arduino-IDE，基于Arduino这点和MWC飞控是一样的。功能上依然很强大，地面站使用apm-planner和mission-planner。最新版本为APM 2.6，开源。

PX4是apm的升级版，使用了很流行的stm32f407单片机，处理速度上了一个大台阶，摆脱了arduino的瓶颈，走向了发烧级的道路。不过PX4实际上是由两部分组成的，PX4FMU和PX4IO板。



后来又推出了Pixhawk飞控，其实就是把PX4FMU和PX4IO板结合在了一起，更高的集成度。



其背后的商业公司是3D Robotics。3D Robotics创始人是克里斯·安德森（Chris Anderson），前《连线》主编，著有《长尾理论》，《创客：新工业革命》等畅销书。创客运动的积极倡导者和领袖级人物。有关克里斯·安德森和他的无人机，可以参看《开源杂志》上的报道《变革推动者.克里斯安德森的无人机梦想腾飞》.

围绕着3DR公司的产品，形成了一个活跃的无人机爱好者社区，无人机飞控功能很强大，能飞固定翼、旋翼、直升机等。旗下网站也有好几个ardupilot.com、diydrones.com、3drobotics.com，很是混杂，很多资料和文档看的眼花缭乱。

据业内人士透露，diydrones是现在最活跃的四轴飞行器，无人机论坛，日活跃用户有2万多（2014-6，这个数据是否真实没有验证过）。所以把这个放在第一位介绍给大家。

- 3DR官网： http://3drobotics.com/。这个网站简单来说就是3dr卖飞控的官方旗舰店，我还以为开发者的一些东西也在上面，，好浪费感情。
- apm飞控的官网：http://copter.ardupilot.com 。见名知意，基于arduino的飞控，肯定是apm的官网了，一些开发者资料在http://dev.ardupilot.com。
- pixhawk飞控的官网：https://pixhawk.org 。 PIX4和Pixhawk的资料都在上面，此外上述代码都托管在GitHub上。
- diydones论坛：http://diydrones.com 。这个是3dr旗下的交流论坛，你可以在这里找到很多国外的飞控爱好者。

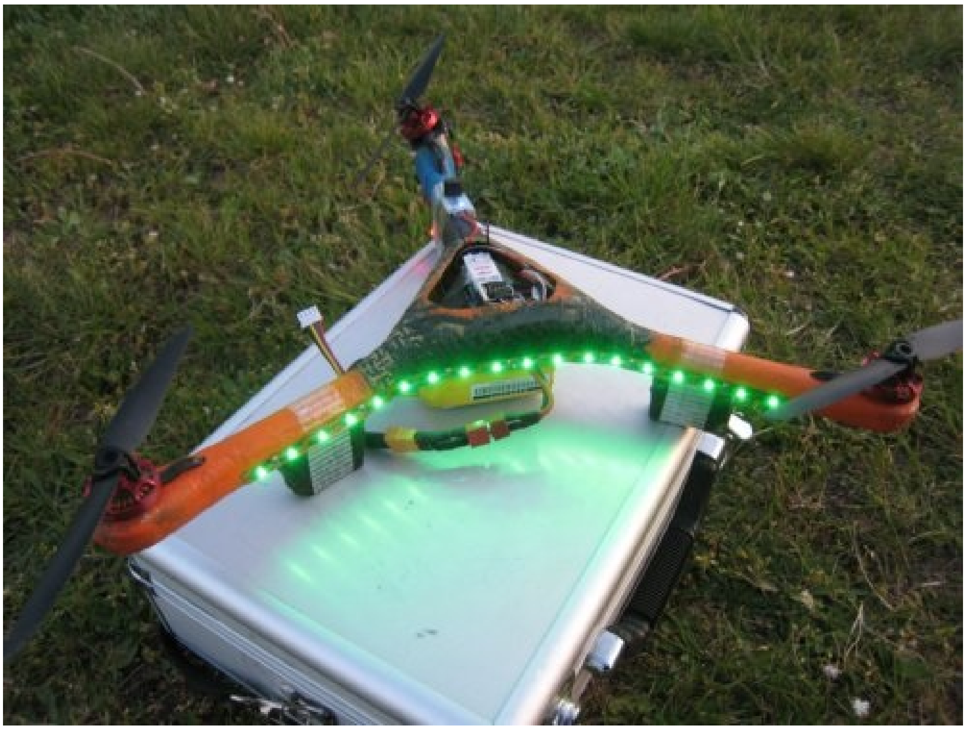


本节内容部分来自Wellmakers博客3D Robotics相关介绍。

MWC & MultiWii

MWC是MultiWii Copter的缩写，是最为经典的开源的多轴飞行器固件。此固件的原创作者是法国的Alex，他为了打造自己的Y3飞行器（一个三轴飞行器），在开源硬件平台Arduino的基础上开发了最初的MWC固件。几年来经过许多高手的参与及共同努力，开发进度越来越快，已经基本成熟，支持三轴，四轴，六轴等多种飞行器。其最大的特点是，其硬件是基于**Arduino**平台。这为很多熟悉开源硬件平台Arduino的玩家入门提供方便。

下图为Alex最早使用MWC的Y3飞行器。



根据使用Arduino开发板的不同，MWC飞控有两种版本。

- Atmega328P 版本，32K的Flash，Atmel公司8位AVR单片机内核
- Atmega2560 版本，256K的Flash，Atmel公司8位AVR单片机内核

> AVR单片机内核就是AVR内核，和51内核是不一样的，它的汇编指令与51单片机汇编指令是完全不一样的。AVR内核是美国ATMEL公司研发的。ATMEL公司有三大系列MCU，一种是老式的以8051内核的单片机，第二种是目前大量使用的以AVR内核的AVR单片机。第三种是目前高端的以ARM内核的微处理器。

所以我们可以使用Arduino Pro Mini/Arduino Mega等开发板配合传感器开发板来搭建自己的四轴飞行器。当然，需要有数字电路和编程的底子，不过如果悟性好，看看文档也能搞起来。主要难点在于调试，很难把它调得很稳，需要很大的耐心。

- MultiWii官网：http://www.multiwii.com/
- MultiWii官方在Google Code上的代码库：http://code.google.com/p/multiwii/
- MultiWii官方在Github上的代码库：https://github.com/multiwii

> 正是由于MWC使用的是8位单片机做的主控芯片，在后来的发展中受到了性能的限制。很多玩家就把MWC飞控代码移植到32位处理器上（例如STM32），形成了BaseFlight飞控代码，后来从BaseFlight项目又分出来（fork）了CleanFlight飞控项目。

CC3D & OpenPilot

OpenPilot社区主要提供下面两个版本的飞控硬件平台：

- OpenPilot CopterControl Platform（又叫CC3D平台）。使用STM32F1作为主控，没有气压计，价格便宜，功能稳定，已经被大量使用和接受。
- OpenPilot Revolution Platform（还在开发中，未发布，2015-5-2）。使用STM32F4作为主控。



配合上面两个飞控硬件平台，OpenPilot还提供一个功能强大，支持跨平台（Windows，Mac OS，Linux）的地面站（包含上位机功能）。



CC3D是现在OpenPilot现在提供的唯一大量使用的飞控。由于其飞控固件稳定，配置简单，价格便宜，而且有强大的地面站上位机支持，所以非常受欢迎。穿越机**QAV250**就是使用的**CC3D**作为默认飞控，crazepony团队会花很多精力来研究这个飞控，以便给大家提供更好的技术支持。

> OpenPilot本身提供的飞控固件已经非常稳定。但是很多穿越机的玩家会把CleanFlight飞控固件烧入到CC3D飞控平台上，以获得更好的飞行体验。更多内容，请看CleanFlight飞控固件部分。

- OpenPilot官网：www.openpilot.org
- OpenPilot Wiki：https://wiki.openpilot.org

Cleanflight

在前面谈到MWC和CC3D飞控的时候，都有谈到Cleanflight，那么到底什么是Cleanflight呢？在crazepony的wiki中为什么要详细单独介绍它呢？

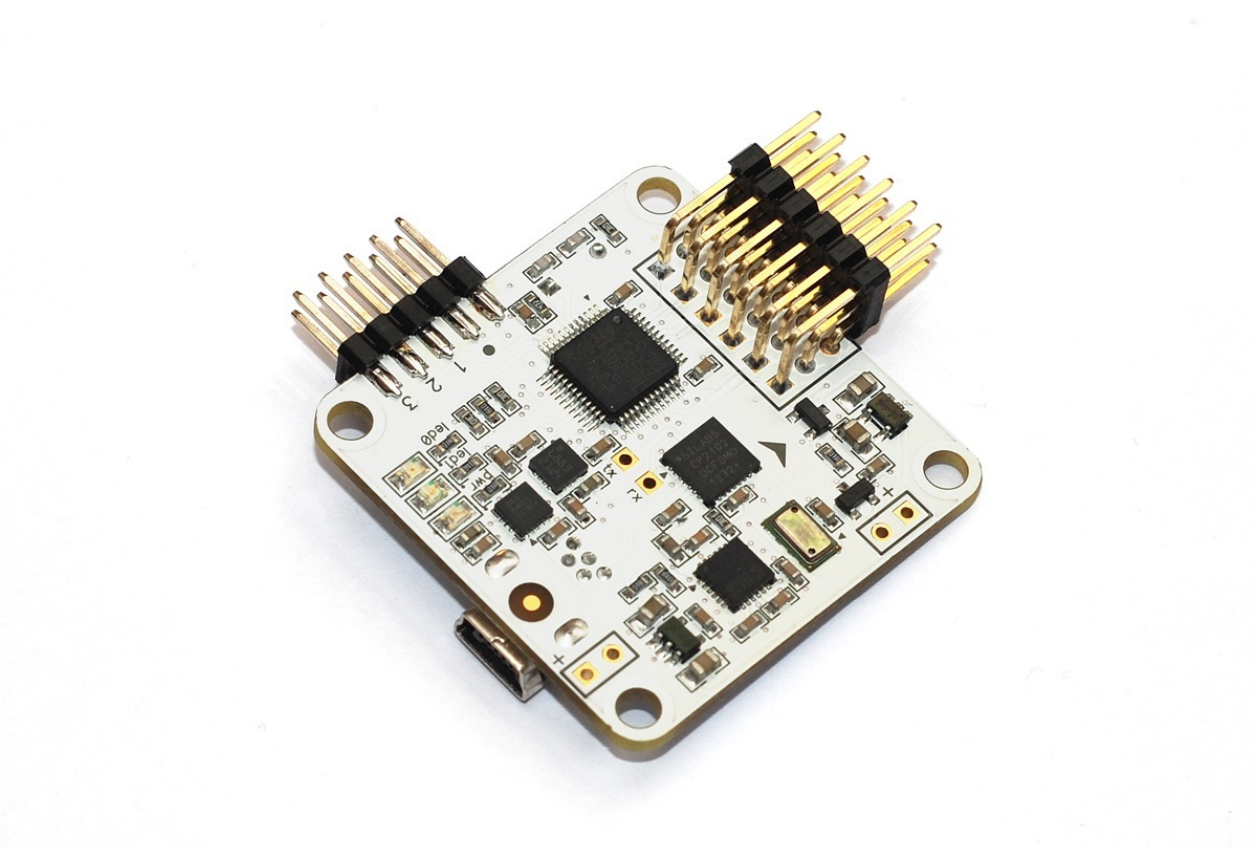
正如在MWC介绍中提到的，由于MWC使用的是8位单片机做的主控芯片，在后来的发展中受到了性能的限制。很多玩家就把MWC飞控代码移植到32位处理器上（例如STM32），形成了BaseFlight飞控代码，后来从BaseFlight项目又分出来（fork）了CleanFlight飞控项目。Cleanflight其实只是一个基于MWC的飞控代码，它现在已经支持了下面这些飞控硬件平台：

- AlienWii32
- Naze32
- CC3D
- CJMCU
- CheBuzzF3
- Olimexino
- Sparky

上面的硬件飞控平台，除了CC3D比较有名之外，Naze32算是一个后起之秀。但是他们都有一个特点，基本上都是基于STM32主控MCU，内核为32位ARM Contex-M。那么crazepony的wiki中为什么要详细介绍这个飞控呢，有下面几点：

- crazepony也是使用的STM32作为主控MCU，而且硬件选型和Naze32完全一样，很有借鉴意义
- CC3D和Naze32飞控都可以刷Cleanflight的飞控，用于研究CrazeFPV H250穿越机
- Naze32作为后起之秀，有超越CC3D的趋势，在国外穿越机玩家中非常流行

Naze32的正面图，和CC3D最大的区别是它有加入气压计MS5611：



下面是Cleanflight的官网和代码库。

- Cleanflight官网：http://cleanflight.com/
- Cleanflight官方在Github上的代码库：https://github.com/cleanflight

cleanflight飞控几个大的特点：

- 飞控代码基于MWC，稳定简洁；
- 为现在的32位处理器而开发，删除了原来为了支持8位处理器而存在的冗余代码。
- 加入了很多好玩的功能。例如支持LED灯带，支持OLED，支持超声波传感器
- 使用了先进的代码开发管理，代码结构清晰，冗余性强，测试充分
- 支持多个硬件平台，例如CC3D，Naze32等

MikroKopter

在2006年10月24号，Holger Buss和Ingo Busker创造了MK，一个伟大的Mikrokopter四轴社区。 在2007年中，Mikrokopter便像一个“ 空中的钉子”，像一只鸟一样，稳步的停留在空中。这对于开源四轴飞行器是一个很大的里程碑。

MikroKopter来自德国。毫无疑问，MikroKopter这个名字来源于德语，对应的英文应该是MicroCopter。其英文官网地址是http://www.mikrokopter.de/en/home。

作者在SVN上共享了代码。



KK飞控

最经典的多轴飞控，价格十分便宜，很多人就是用它学会飞四轴的。主要问题是没有自稳，更不用说什么定高、GPS之类的高级功能了。因为没有自稳，新手练习起来就比较困难，不过对于从直升机转过来的玩家来说就是小case了。支持3轴、四轴、6轴、V22的飞行模式；

http://www.kkmulticopter.com/

Paparazzi

这是国外一个开源的固定翼，多轴飞行器项目。一个做固定翼的朋友购买了它的组件。github上的资料，wiki等都挺全面的。项目不仅仅只包括飞控板的软硬件，包括从稳压电源和GPS到卡尔曼滤波代码，而是一个强大和不断扩大的地面空中软硬件群，包括数传电台，天线和一个高度进化的用户友好的地面控制软件界面。

http://wiki.paparazziuav.org/wiki/Main_Page

https://github.com/paparazzi/

open drone（国内）

http://www.open-drone.org/

https://github.com/opendrone

open drone是由北京的LUG和创客组建的一个项目。但是没有看到其产品。很可能还是一个比较松散的组织，没有以公司盈利的团队形式存在。

erlerobot

这是一个使用Linux系统的开源四轴飞行器项目。他们的口号是：Building the next generation of educational drones。

http://erlerobot.com/

https://github.com/erlerobot

有比较齐全的wiki，并且使用gitbook形式提供四轴飞行器开发的电子书。

https://github.com/erlerobot/wiki/wiki

http://erlerobot.github.io/erle_gitbook/en/

四轴飞行器明星产品

> 作者：nieyong

Phantom





Phantom小型四轴航拍飞行器是专门针对想接触航拍，但又苦于没有经验、担心太复杂的用户而设计，整体解决方案包含飞行器、自动控制、动力、遥控等等系统，用户购买后只需要自配一部相机，进行简单的组合即可享受航拍的乐趣，免除自行DIY拼装、调试等等繁复步骤。

Phantom是中国公司大疆创新推出的明星产品。现在已经推出了最新的Phantom vision+。

AR.Drone



通过WiFi链接iPhone或者iPad或者iPod Touch，在一定范围内进行遥控飞行。这个飞行器有四个螺旋桨，可以选择四个环形对螺旋桨进行保护。飞行器提供任何飞行角度，机体前方和下方带有摄像头，拍摄画面实时传送。

此外，飞行器还具有重力感应、陀螺仪等部件，拥有智能飞行技术，自动矫正风力和环境误差，看上去很是吸引人,更为玩家们惊喜的是，2台以上Ar.Drone可模拟空战游戏，使得Parrot AR.Drone完全不在等同于一般的航模遥控飞机，而玩家只需要到苹果App Store中下载相关软件即可。

AR.Drone是法国公司Parrot的明星产品。现在已经出到第三代。

直升机 VS 多旋翼 VS 固定翼

> 作者：nieyong

航模飞行器，最常见的有直升机，多旋翼，固定翼。它们有各自的特点和支持者，同时又是相互联系。掌握了其中的一种就可以融会贯通，举一反三。

直升机

直升机航模是航模中最为资深的，也是对操控技术要求最高的一类。非常适合练习飞手的操控技能，能够飞出各种动作，表演具有很强的观赏性。

现在很多航模比赛也是以直升机类最为受关注和技术含量最高。



多旋翼

多旋翼也就是我们看到的四轴，六轴等飞行器。是最近几年兴起的一类航模飞行器。多旋翼操作简单，入门门槛低，航模爱好者多用于航拍。

最近流行使用多旋翼进行第一视角FPV竞速比赛，而且发展势头非常迅猛。



固定翼

固定翼飞行器主要需要考虑飞行器的空气动力特点。续航时间长，不能够垂直起飞。



大四轴 VS 小四轴

> 作者：nieyong

按照四轴飞行器的大小，我们粗略的将四轴飞行器分为大四轴和小四轴。其有很多部分是相同的，例如飞控部分基本一样；但是又有区别，例如动力部分（包括电机和电机驱动）就可能完全不相同。作为四轴飞行器爱好者，我们当然需要知道DIY大小四轴的区别。

当然，Crazepony是一个小四轴。

大四轴

大四轴根据其承载能力的大小，也分为不同的级别。例如DJI的S800大四轴，能够挂载专业的单反相机，主要用于电影等的拍摄，其大小有大人双手怀抱那么大。风火轮系列则要小很多，大概有脸盆大小，适合挂载gopro等运动相机。



对于航模爱好者来说，DIY一个大的四轴飞行器也不是一件容易的事情。你可以购买品牌厂商的整套解决方案，然后自己进行组装调试，例如DJI的S800系列。不过对于很多爱好者来说，价格会比较贵，一套S800的价格在3万人民币左右。你也可以自己购买组件，自己组装。一般需要购买的组件包括：飞控，电调，无刷电机，遥控器，电池，支架等。当然，也见过有爱好者自己用各种材料DIY支架，有用铝合金的，也有用木头的。根据自己的需求购买组件DIY的成本会低很多，一般的可能就是￥1000～￥2000左右。

DIY大四轴需要有比较专业的知识，例如电子部分的连接，飞行器结构重心的调整，遥控器的对频和干扰，根据飞行器结构或者飞行现场风力等对飞控PID等参数的调节（简称调参）。总之，如果有一个环节出现了问题，那么就会导致飞行器无法飞起来，或者出现炸机（指飞行器从空中摔下来）或者失控（指飞行器不受遥控器控制）的问题。



另外还需要注意的一点是，大四轴由于其块头大，螺旋桨转速快，撞到人或者物都会比较危险。所以一定要选择无人，空间宽阔，视野好的地方进行飞行。常见的一个事故就是射桨。射桨就是飞行器的螺旋桨因为旋转时转速太高，超出了旋翼所承受拉力的设计值时，桨根断裂，由于旋转时的巨大惯性，残桨像炮弹破片那样高速飞出去，威力很大（相对来说），会打伤人，所以形象的称为“射桨”。有时旋翼固定螺丝松脱或碰到异物导致旋翼断裂也会导致射桨。如果射桨击中人，尤其是眼睛等脆弱的部位，会是非常危险的。所以对于新手来说，调四轴最好带护目镜保护眼睛。

小四轴

小四轴是相对于上面的大四轴来说的，这里我们指手掌大小的四轴飞行器。一般来说，小四轴直接使用电路板作为承载的支架，当然也有使用单独支架的。

小四轴由于其体型小，所以一般无法承载太重的负载。但是用来研究飞行器的原理，作为入门航模的通道，已经完全绰绰有余。在大四轴中提到的组件飞控，电调，电机，遥控器，电池，支架等都一样在小四轴中存在，只是可能根据小四轴的特点有些改变。下面这些就是小四轴的好处。

- 价格便宜，相对于大四轴动则上千元的成本，小四轴一般几百块钱就能够搞定
- 危险性很小，基本上随处都能飞
- 自己设计飞控电路板，自己移植编写开源飞控程序和算法。而大四轴的飞控一般是商业

公司的产品，开源有限，DIY的可能性少

- 由于小四轴飞行的距离有限，所以可以省去遥控器，直接使用智能手机控制

航模基本概念扫盲

人类从来就有一个飞行的梦想，从风筝到竹蜻蜓，从热气球到莱特兄弟，从空客A380到航天飞船。也许这个梦想就是从孩子手中小小的航模飞机开始种下的种子，等待某一天梦想的实现。

最基本，最原始的，那才是以后腾飞最重要的助推剂。本章将会为你介绍航模中最基本最常见的概念。



航模常用术语

> 作者：nieyong

航模术语部分涉及到大四轴。如果大小四轴有区别的，会特别说明，谁叫我们Crazepony是小四轴呢。

飞控篇

- 飞控：飞控指飞行器的电子控制部分，硬件包括传感器部分惯性导航模块和控制部分的MCU。软件包括控制算法等。
- **pitch，yaw，roll**：指三维空间中飞行器的旋转状态，对应中文分别是俯仰，航向，横滚。
- 惯性导航模块：简称IMU，陀螺仪传感器和加速度传感器提供的三轴运动数据模块。
- 运动感测追踪：英文Motion Tracking。
- 地理坐标系：指地球所在的坐标系，这个坐标系是固定不变的，正北，正东，正上方分别表示X，Y，Z轴。
- 姿态解算：英文attitude algorithm，也叫做姿态融合。姿态解算是指把陀螺仪、加速度计、罗盘等的数据融合在一起，得出飞行器的空中姿态。
- 深度解算：也叫做长期融合。
- 快速结算：也叫做快速融合。
- 四元数：

桨叶篇

- 射桨：飞行器的螺旋桨因为旋转时转速太高，超出了旋翼所承受拉力的设计值时，桨根断裂，由于旋转时的巨大惯性，残桨像炮弹破片那样高速飞出去，威力很大（相对来说），会打伤人，所以形象的称为“射桨”。有时旋翼固定螺丝松脱或碰到异物导致旋翼断裂也会导致射桨。

电机电调篇

- 有刷/无刷电机：有刷电动机是内部含有换相电刷的电动机，无刷电机则不需要电刷。
- 空心杯电机：coreless motor，属于直流、永磁、伺服微特电机。空心杯电机在结构上突破了传统电机的转子结构形式，采用的是无铁芯转子。
- 电调：电调全称叫做电子调速器，英文electronic speed controller，简称ESC。用于控制电机的转动，停止，以及转动速度。

遥控器篇

电池充电器篇

- 航模电池：航模上使用的锂电池，需要有较高的放电能力，一般以单位C表示，这是普通锂电池和航模锂电池最重要的区别。例如Crazepony就是使用25C的350mAh锂电池。
- 电池容量**mAh**：表示电池容量，如1000mah电池，如果以1000ma放电，可持续放电1小时。如果以500mh放电，可以持续放电2小时。
- 电池节数**2s/3s/4s**：代表锂电池的节数，锂电池1节标准电压为3.7v，那么2s电池，就是代表有2个3.7v电池在里面，电压为7.4v。对于小四轴，一般就是使用一节3.7v电池就可以。
- 平衡充电：对于多节组合的电池，如3s电池，内部是3个锂电池，因为制造工艺原因，没办法保证每个电池完全一致，充电放电特性都有差异，电池串联的情况下，就容易造成某些放电过度或充电过度，充电不饱满等，所以解决办法是分别对内部单节电池充电。动力锂电都有2组线，1组是输出线（2根），1组是单节锂电引出线（与s数有关），充电时按说明书，都插入充电器内，就可以进行平衡充电。对于只有一节电池的小四轴，不存在这个平衡充电问题。

其它

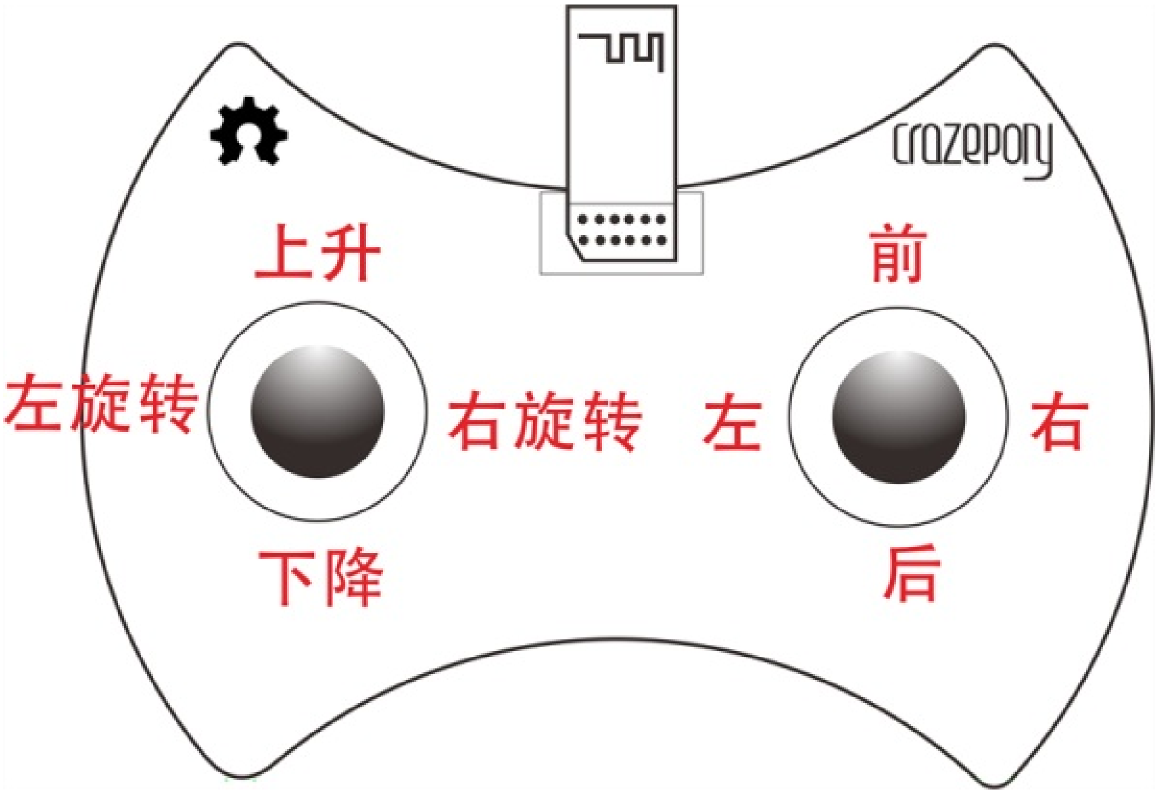
部分内容参考了Taculee博客四轴飞行器基础入门。

美国手 VS 日本手

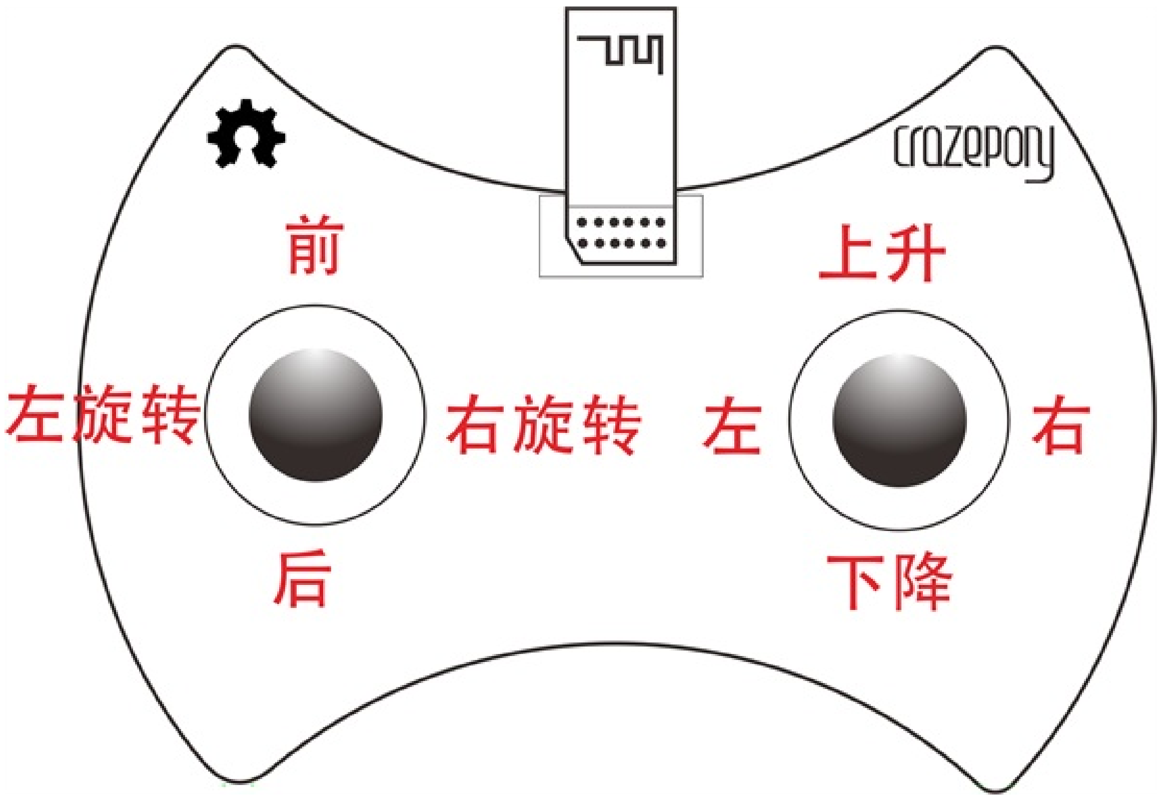
> 作者：nieyong

航模遥控器根据操控方式不一样，分为美国手和日本手。

美国手也就是说左手摇杆负责油门和偏航，右手摇杆负责俯仰和横滚。左手摇杆上下为油门控制，左右是偏航控制；右手摇杆上下为俯仰控制，左右为横滚控制。



对应的另外一种遥控为日本手。左手摇杆负责俯仰和偏航，右手摇杆负责油门和横滚。是不是很别扭？将俯仰和横滚的控制分离开这种设计太反人类了个人觉得，但是它就是这样一个标准没办法，日本人通常都会做些我们难以理解的事大家都知道的。



> Crazepony遥控默认固件为美国手固件。修改为日本手遥控只需要重刷遥控固件就可以。

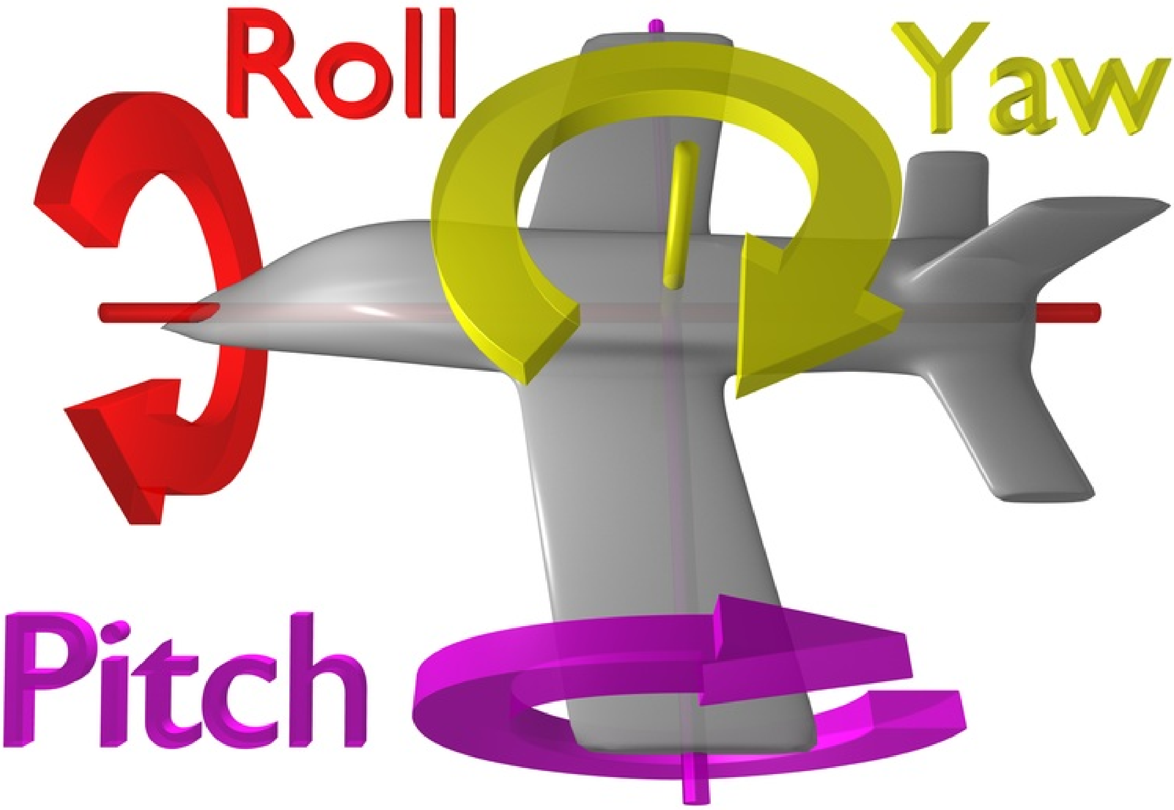
为什么会有日本手和美国手呢？你到底是选择哪种遥控方式呢？下面这段有趣的文字摘自潘农菲（大疆副总裁）的《坚持不做手机操控！只因简单的两个字》。

> 玩航模的朋友都知道美国手和日本手，美国手就是左手油门，右手前进后退，日本手不同（左手：上下控制飞行器的前进后退，左右控制左右旋转机头。右手：上下控制飞行器上升下降，左右控制飞行器左右平移)。
>
> 有次，我和一个培训出很多世界冠军的香港飞手聊，为什么很多专业的飞手都是日本手？
>
> 他回答，除了早年设备都是日本手这个原因外，还有一个原因：用并不惯用的左手控制前进后退，左手的天生弱项反而可以让飞手通过训练更加精细的操控，因此很多顶级高手使用这种方式。
>
> 这让我想到多年前我学日本剑道时，问一名台湾五段，为什么要用相对没有力量的左手持剑，而更有力量的右手仅仅做辅助？
>
> 他回答，右手是感情，左手是理智，剑是危险的事物，要用理智来操控。
>
> 虽然有些玄虚，但是这句话我倒是记住了。
>
> 多年后当我学无人机飞行操控时，这段话竟然做出了最精准的解释。

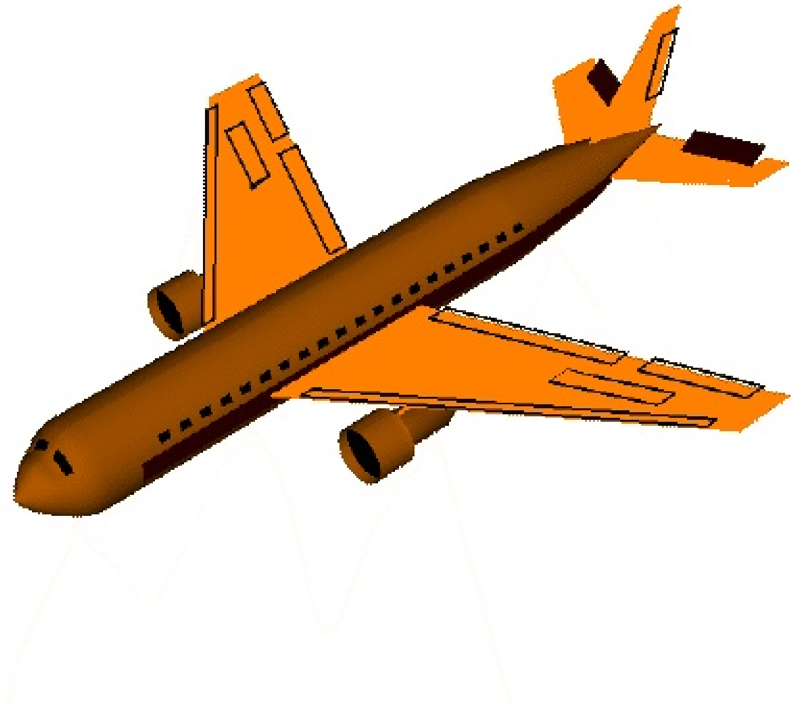
三维中pitch，yaw，roll的区别

> 作者：nieyong

玩过航模，飞控的童鞋肯定需要知道 yaw, pitch and roll 的区别。这里给大家分享一下，准备要做飞控的，或者想玩三轴加速度的，或者玩了很久加速度还没研究过哪个是哪个的，可以看一下。



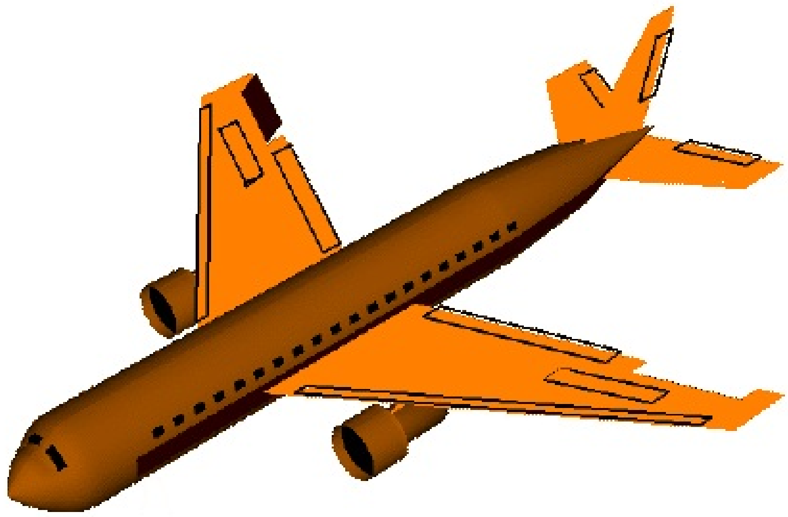
pitch()：俯仰，将物体绕X轴旋转（localRotationX）



yaw()：航向，将物体绕Y轴旋转（localRotationY）



roll()：横滚，将物体绕Z轴旋转（localRotationZ）



四轴飞行器的空气动力原理

> 作者：nieyong

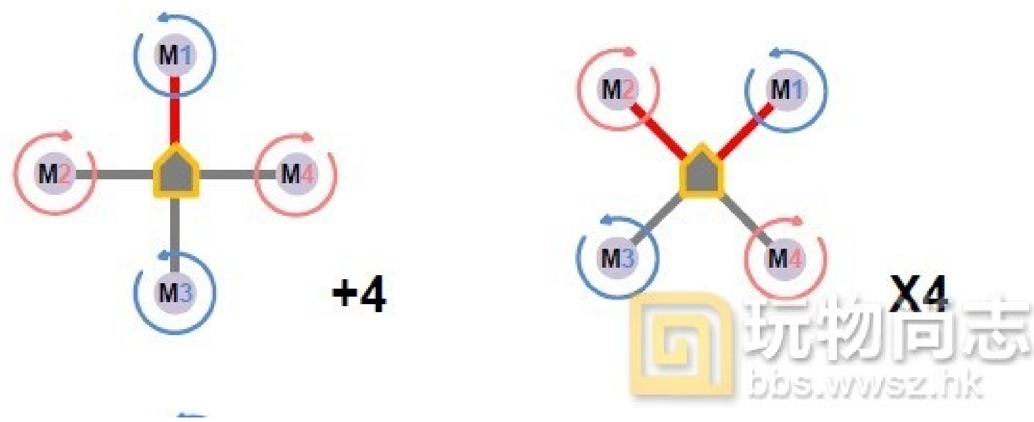
四旋翼飞行器结构以及x模式和+模式

四轴飞行器四个电机呈十字形排列, 驱动四片桨旋转产生向上的推力。四个电机轴距几何中心的距离相等, 当对角两个轴产生的升力相同时能够保证力矩的平衡, 四轴不会向任何一个方向倾转; 而四个电机一对正转,一对反转的方式使得绕竖直轴方向旋转的反扭矩平衡, 保证了四轴航向的稳定。

与传统的直升机相比，四旋翼飞行器有下列优势：各个旋翼对机身所施加的反扭矩与旋翼的旋转方向相反，因此当电机1和电机3逆时针旋转的同时，电机2和电机4顺时针旋转，可以平衡旋翼对机身的反扭矩。

根据用户自定义的机头的位置不同，四轴飞行器可以分为x模式和+模式。x模式的机头方向位于两个电机之间，而+模式的机头方向位于某一个电机上。x和+就是表示正对机头方向时飞行器的形状。如下图所示。x模式要难飞一点，但动作更灵活。+模式要好飞一点，动作灵活差一点，所以适合初学者。特别注意，x模式和+模式的飞控安装是不同的。如果飞控板安装错误，会剧烈的晃动，根本无法飞。

crazepony使用的是**x**模式。



空气动力原理

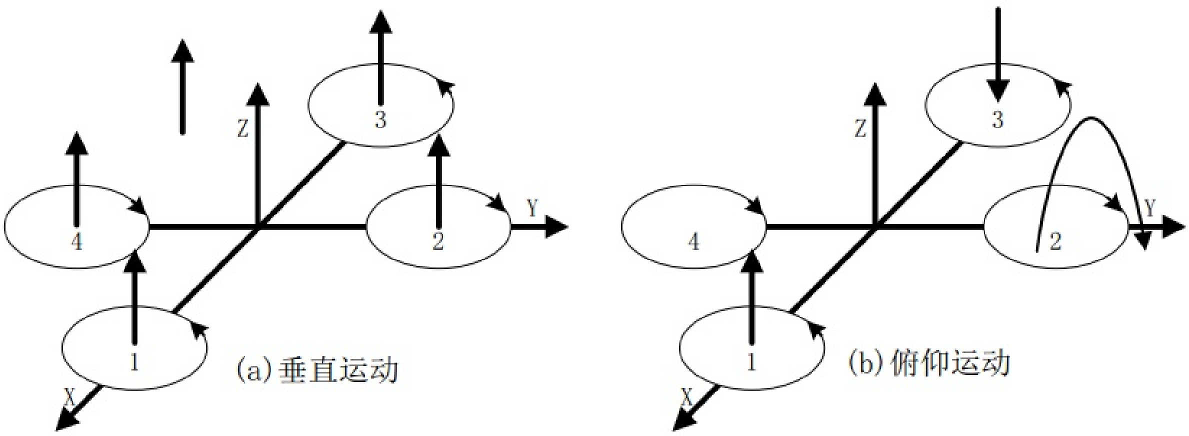
四旋翼飞行器在空间共有6个自由度（分别沿3个坐标轴作平移和旋转动作），这6个自由度的控制都可以通过调节不同电机的转速来实现。基本运动状态分别是：

- 垂直运动；
- 俯仰运动；
- 滚转运动；
- 偏航运动；
- 前后运动；
- 侧向运动。

在图中，电机1和电机3作逆时针旋转，电机2和电机4作顺时针旋转，规定沿x轴正方向运动称为向前运动，箭头在旋翼的运动平面上方表示此电机转速提高，在下方表示此电机转速下降。

垂直运动：垂直运动相对来说比较容易。在图中，因有两对电机转向相反，可以平衡其对机身的反扭矩，当同时增加四个电机的输出功率，旋翼转速增加使得总的拉力增大，当总拉力足以克服整机的重量时，四旋翼飞行器便离地垂直上升；反之，同时减小四个电机的输出功率，四旋翼飞行器则垂直下降，直至平衡落地，实现了沿z轴的垂直运动。当外界扰动量为零时，在旋翼产生的升力等于飞行器的自重时，飞行器便保持悬停状态。保证四个旋翼转速同步增加或减小是垂直运动的关键。

俯仰运动：在图（b）中，电机1的转速上升，电机3的转速下降，电机2、电机4的转速保持不变。为了不因为旋翼转速的改变引起四旋翼飞行器整体扭矩及总拉力改变，旋翼1与旋翼3转速该变量的大小应相等。由于旋翼1的升力上升，旋翼3的升力下降，产生的不平衡力矩使机身绕y轴旋转（方向如图所示），同理，当电机1的转速下降，电机3的转速上升，机身便绕y轴向另一个方向旋转，实现飞行器的俯仰运动。

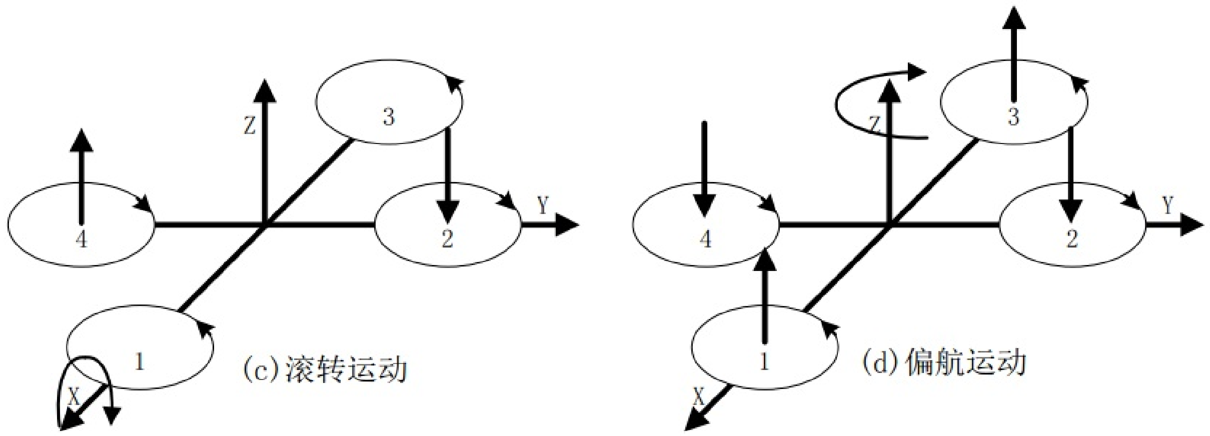


(a)垂直运动 (b)俯仰运动

滚转运动：与图b的原理相同，在图c中，改变电机2和电机4的转速，保持电机1和电机3的转速不变，则可使机身绕x轴旋转（正向和反向），实现飞行器的滚转运动。

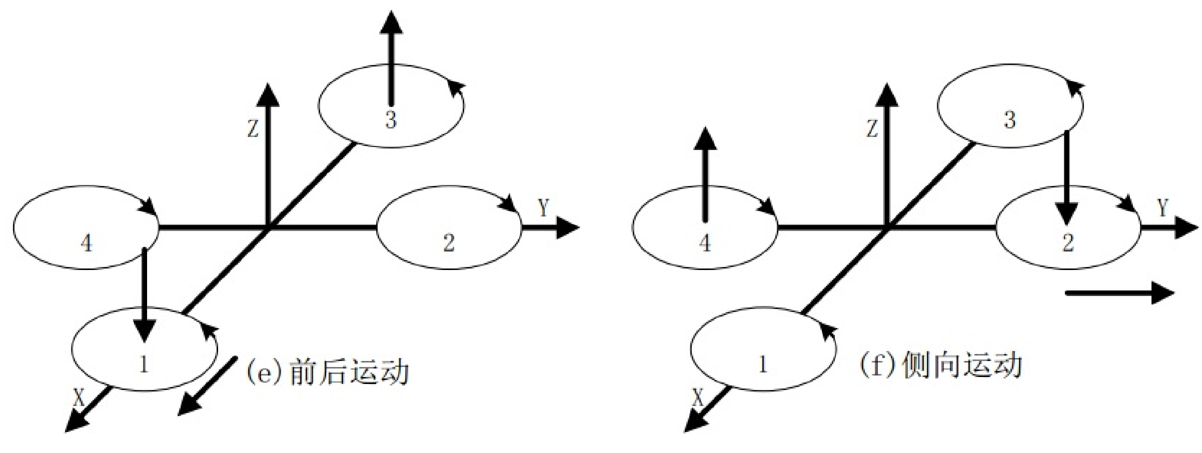
偏航运动：四旋翼飞行器偏航运动可以借助旋翼产生的反扭矩来实现。旋翼转动过程中由于空气阻力作用会形成与转动方向相反的反扭矩，为了克服反扭矩影响，可使四个旋翼中的两个正转，两个反转，且对角线上的各个旋翼转动方向相同。反扭矩的大小与旋翼转速有关，当四个电机转速相同时，四个旋翼产生的反扭矩相互平衡，四旋翼飞行器不发生转动；当四个电机转速不完全相同时，不平衡的反扭矩会引起四旋翼飞行器转动。在图d中，当电机1和

电机3的转速上升，电机2和电机4的转速下降时，旋翼1和旋翼3对机身的反扭矩大于旋翼2和旋翼4对机身的反扭矩，机身便在富余反扭矩的作用下绕z轴转动，实现飞行器的偏航运动，转向与电机1、电机3的转向相反。



前后运动：要想实现飞行器在水平面内前后、左右的运动，必须在水平面内对飞行器施加一定的力。在图e中，增加电机3转速，使拉力增大，相应减小电机1转速，使拉力减小，同时保持其它两个电机转速不变，反扭矩仍然要保持平衡。按图b的理论，飞行器首先发生一定程度的倾斜，从而使旋翼拉力产生水平分量，因此可以实现飞行器的前飞运动。向后飞行与向前飞行正好相反。当然在图b图c中，飞行器在产生俯仰、翻滚运动的同时也会产生沿x、y轴的水平运动。

倾向运动：在图f中，由于结构对称，所以倾向飞行的工作原理与前后运动完全一样。



总得来说就是控制四个电机的速度了。然后相应的提高速度和减慢速度就可以让四轴动起来了。

飞行器构造公开课

麻省理工学院公开课：飞行器构造工程，来自网易公开课。讲师Prof. Jeffrey Hoffman,Prof. Aaron Cohen，教授，曾供职于美国国家航空航天局（NASA）。

无头模式 VS 有头mohican

> 作者：nieyong

飞行器在飞行的过程中，其运动的前后左右以地理坐标系为参考坐标系，则为无头模式（又叫head-free或head-hold）飞行。无头模式多见于多旋翼，例如四轴，六轴飞行器中。

相对应的就是有头模式，飞行器运动的前后左右以自身的坐标系为参考坐标系。有头模式是很多玩具航模，大多旋翼飞行器（例如Phantom）常用的飞行模式。

无头模式 & 有头模式

这里有必要对上面的定义做展开讲解。任何飞行器都一定有个自身的坐标系，也就是飞行器的头，尾，这也就是前面说的飞行器的自身坐标系。如果推动遥控器的向前运动，飞行器总是向它头的方向飞行，那么这个飞行器就是运行在有头模式。如果推动遥控器的向前飞行，飞行器还是向它起飞时头指示的方向飞行，即使这个时候飞行器在飞行的过程中改变了机头方向（操纵了遥控的航向角），那么 这个飞行器的运行在无头模式。

所以可以看到，无头模式飞行器是以它起飞时头指示的方向为前，其运行的前后左右是参考地理坐标系（也就是地球坐标系）。所以飞无头模式，操控手尽量不要旋转，否则就会混淆运行的前后左右。

无头模式的乐趣

飞行过程中，无需考虑机头的方向，对于速度飞行会更爽。

如何实现无头模式

前面有说到，飞行器无头模式是以地理坐标系，也就是地球坐标系为参考，所以最常见的实现，就是使用电子罗盘，测量飞行器相对于地球磁场的角度，算出机头在磁场中的方向。

另外，就是使用航向角的积分，计算飞行器相对于起飞时机头旋转的角度。这就是crazepony无头模式的实现。有关无头模式算法的实现，参考todo。

> Crazepony默认固件为无头模式。

四轴飞行器基本组成部分

本章将会为讲解四轴飞行器的基本组成部分，很多都争对Crazepony迷你四轴飞行器做了特殊的说明。但是很多基本组成部分，大小四轴都是相同的。按照模块划分，主要有下面几个部分。

- 动力部分：电机，电机驱动，桨叶
- 控制部分：MCU，传感器（IMU），遥控通信（2.4G和蓝牙）



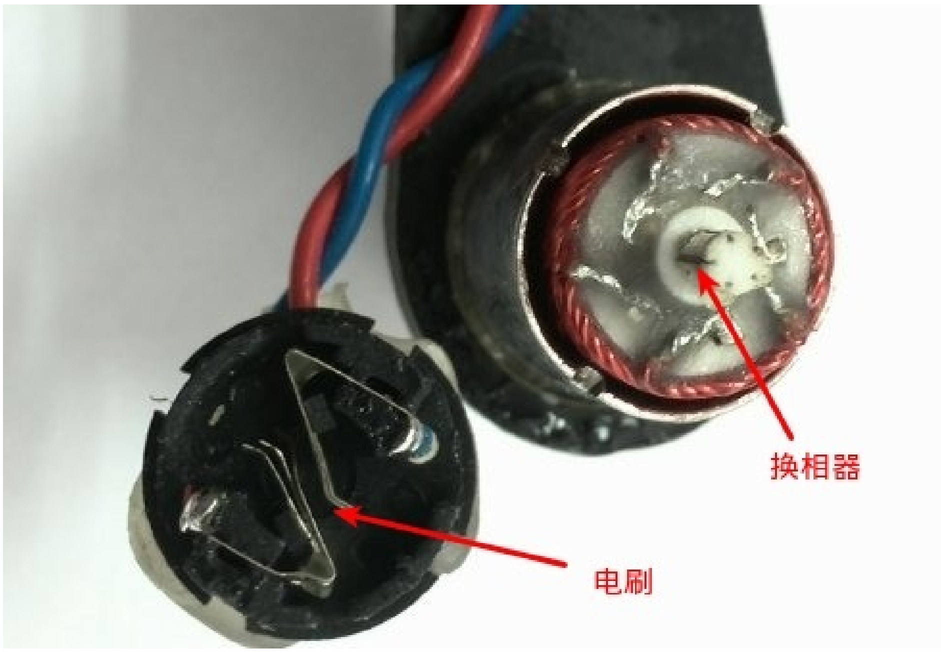
航模电机

> 作者：nieyong

电动机有有刷和无刷之分。结构上，无刷电机和有刷电机有相似之处，也有转子和定子，只不过和有刷电机的结构相反；有刷电机的转子是线圈绕组，和动力输出轴相连，定子是永磁磁钢；无刷电机的转子是永磁磁钢，连同外壳一起和输出轴相连，定子是绕组线圈，去掉了有刷电机用来交替变换电磁场的换向电刷，故称之为无刷电机（Brushless motor）。

有刷电机

有刷电动机是内部含有换相电刷的电动机。什么是电刷呢，直接上图。电刷通过绝缘座固定在电动机后盖上，将电源的正负极引入到转子的换相器上，而换相器连通了转子上的线圈，线圈极性不断的交替变换与外壳上固定的磁铁形成作用力而转动起来。下图是将现在Crazepony使用的电机拆解之后看到的电刷和换相器。



有刷电机由于电刷和换相器之间的物理接触，所以有摩擦大，发热大，效率低等缺点。但是有刷电机同样具有制造简单，成本低廉的优点，所以现在市面上有刷电机仍然占有很大份额。

无刷电机

有刷电机通过电刷和换相器这种结构设计获得固定方向的磁场作用力而转动起来。无刷电机没有电刷和换相器，它是如何获得固定方向的磁场作用力的呢？简单而言，依靠改变输入到无刷电机定子线圈上的电流波交变频率和波形，在绕组线圈周围形成一个绕电机几何轴心旋转的磁场，这个磁场驱动转子上的永磁磁钢转动，电机就转起来了。

电机的性能和磁钢数量、磁钢磁通强度、电机输入电压大小等因素有关，更与无刷电机的控制性能有很大关系，这就是无刷电机配合的电调需要解决的问题。

无刷电机拥有动力足，寿命长，效率高等优势。所以大四轴都是用无刷电机的，模型无刷电机的外观如下，区别于有刷电机，最明显的就是无刷电机有三根线，并且配合电调使用。



无刷电机最重要的一个参数是KV值，这个数值是无刷电机独有的一个性能参数，是判断无刷电机性能特点的一个重要数据。

> 无刷电机KV值定义为转速/伏特，意思为输入电压增加1伏特，无刷电机空转转速增加的转速值（转/分钟）。总这个定义来看，我们能知道，无刷电机电压的输入与电机空转转速是遵循严格的线性比例关系的。

例如：某无刷电机KV值为1000KV，意思是此电机在1V电压下，每分钟转速为1000转。则在10V电压下，此电机转速为：10V*1000KV=10000转/分，当然这些都是在电机空载的时候。

绕线匝数多的，KV值低，最高输出电流小，但扭力大。

绕线匝数少的，KV值高，最高输出电流大，但扭力小。

这里可以扩展一下，为什么四轴飞行器都是使用的外转子无刷直流电机呢？外转子电机将原来处于中心位置的磁钢做成一片片,贴到了外壳上,电机运行时,是整个外壳在转,而中间的线圈定子不动。外转子无刷直流电机较内转子来说,转子的转动惯量要大很多(因为转子的主要质量都集中在外壳上),所以转速较内转子电机要慢,通常KV值在几百到几千之间,用在航模上可以直接驱动螺旋桨,而省去了机械减速机构。（摘自《无感无刷直流电机之电调设计全攻略》）

电机之空心杯

空心杯电机（coreless motor）属于直流、永磁、伺服微特电机。空心杯电机在结构上突破了传统电机的转子结构形式，采用的是无铁芯转子。空心杯电机具有十分突出的节能、控制和拖动特性。

空心杯电机分为有刷和无刷两种，有刷空心杯电机转子无铁芯，无刷空心杯电机定子无铁芯。

Crazepony使用的是有刷空心杯电机，转速为**30000**转**/**分钟，直径为**7mm**，长度为**20mm**（简称**720**电机），电机轴直径为**1mm**。当然，随着现在生产工艺的进步，现在市面上已经有体积非常小的航模无刷电机，Crazepony也会尝试使用无刷电机。

Crazepony空心杯高速电机拆解图，可以明显看到线圈中间没有铁芯的。





扩展阅读

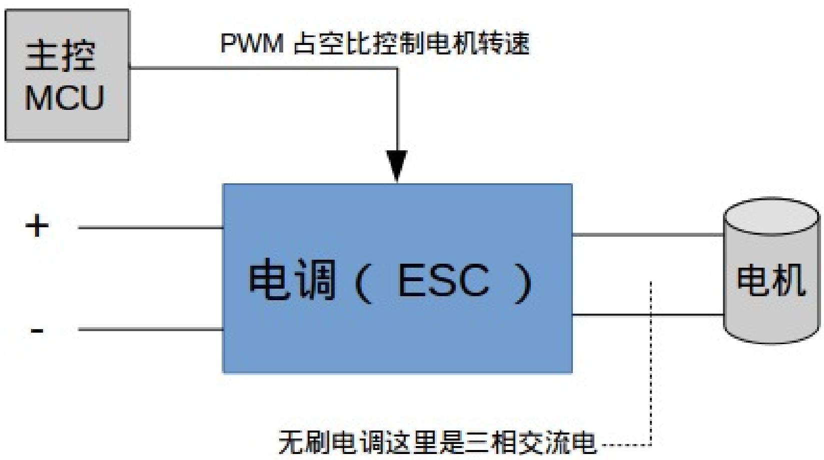
关于无刷直流电机原理的讲解。

航模电机驱动控制

> 作者：nieyong

电机驱动控制就是控制电机的转动或者停止，以及转动的速度。电机驱动控制部分也叫做电子调速器，简称电调，英文electronic speed controller（ESC）。电调对应使用的电机不同，分无刷电调和有刷电调。

有刷电机的永磁体是固定不动的，线圈绕在转子上，通过电刷跟换相器接触来改变磁场方向来保持转子持续转动。无刷电机，顾名思义，这种电机是没有电刷和换相器的，他的转子是永磁体，而线圈是固定不动的，直接接到外部电源，问题就来了，线圈磁场方向怎么改变呢？事实上，无刷电机外部还需要一个电子调速器，这个调速器就是一个电机驱动，通过改变固定线圈内部电流的方向，保证它跟永磁体之间的作用力是相互排斥，持续转动得以延续。



有刷电机工作可以不需要电调，直接把电供给电机就能够工作，但是这样无法控制电机的转速。无刷电机工作必须要有电调，否则是不能转动的。必须通过无刷电调将直流电转化为三相交流电，输给无刷电机才能转动。

一般使用PWM的占空比来控制电机的转速。

Crazepony电机驱动

无刷电机的操作相对来说是比较麻烦的，而有刷电机就是我们小时候玩的四驱车上的那种电机，接上电就能猛转，反着接它就反着猛转，就是这么简单。

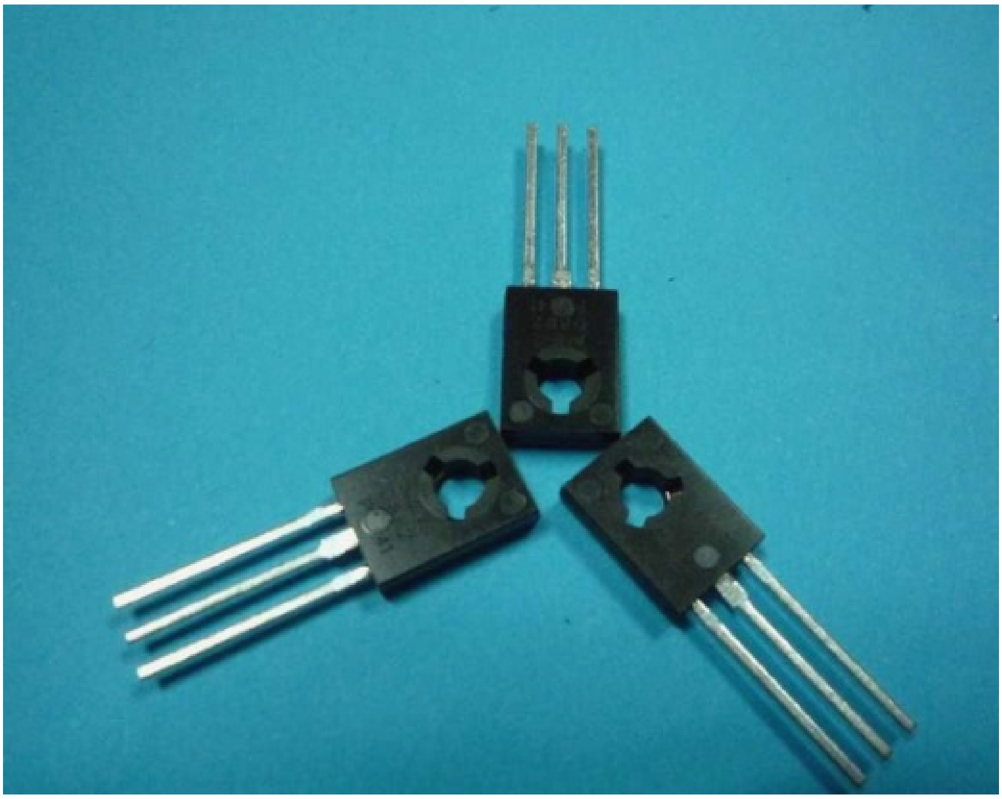
Crazepony使用的是有刷空心杯电机，所以电机的控制属于有刷直流电机控制，相对于无刷电调来说要简单很多。Crazepony采用的是有刷空心杯高速电机，转速在3W转/分钟左右。要驱动有刷电机，很简单，只需要将信号的驱动能力增大，就能驱动有刷电机了。

那么选择什么元件来提供这样的特性呢?Crazepony的电机驱动IC选型经历了三级管，中功率管的失败，最后选用的是场效应管（即MOSFET）SI2302。

由于笔者完全是由于一种强烈的爱好选择了飞行器，最开始连有刷电机和无刷电机的物理结构区别都不知道，电调又是啥？傻傻分不清楚......

> 从一个几乎零基础的状态去选择电机驱动芯片，弯路是必须要走的，学费是必须要交的。曾以为书上学到的东西马上就能用，马上能转化为产品，后来发现真的是自己想多了。

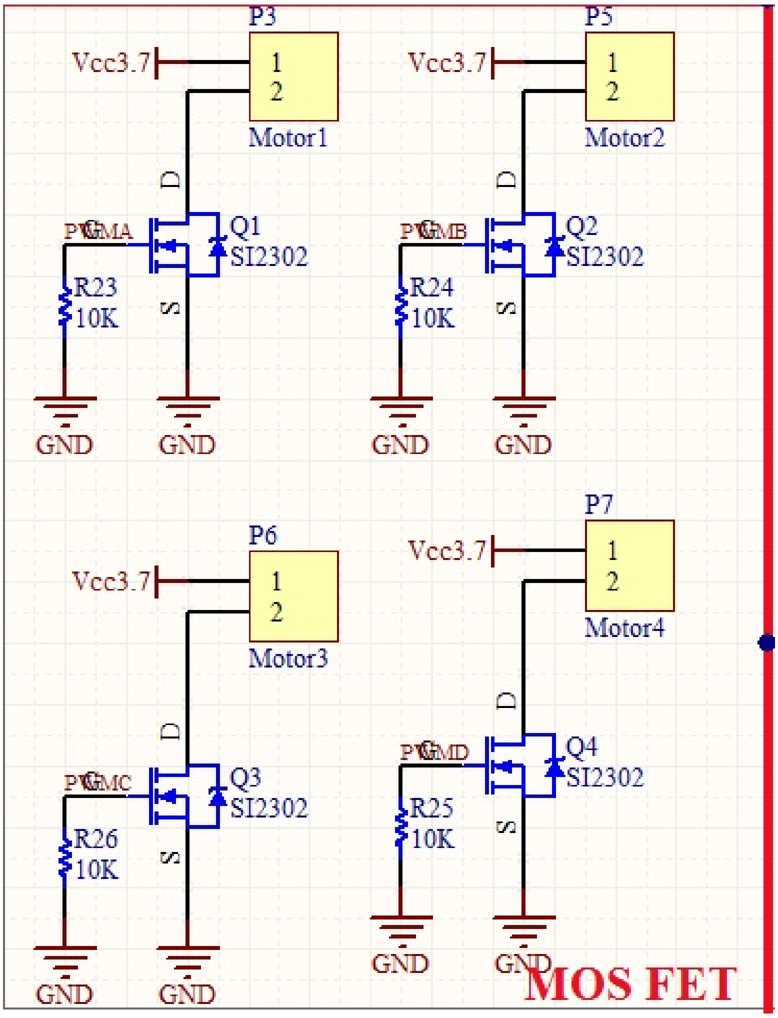
最开始用的三极管作为电机驱动，采用很经典的共射电路“三极管工作在开关状态应该就行了吧？”画了用三极管驱动的PCB板，发现电机越转越慢，根本没劲。“也许是因为三极管扛不了大电流，好吧那我换个中功率管吧，集电极最大6A电流行了吧？”可以想象结果是不行的。



首先了解下为什么三极管作为简单的电机驱动是不可取的方案：

- 三极管作为一个古老的半导体先驱，它是以一个放大器件的姿态而出现的，它在线性区域特性集中，饱和与截止都是两种极端的工作状态，而作为电机驱动的话，我们只能选择它的这两种极端工作模式。
- 用三极管作为大电流负载的驱动管时，不得不考虑的是他自身的管压降对负载的影响，这是很严重的。自身耗散越来越大，电机和管子是串联关系，电池电压只有3.7V，电机就只能越转越慢了

在晶体管家族里面还有一种跟三极管特性互补的，所有特性都集中在开关状态的晶体管，场效应管，即MOSFET。通常的场效应管完全导通时，源漏极电阻都是mΩ级别的，即它自身的耗散非常小。用它做为驱动管再合适不过了。最终选择了一个SOT23封装的,导通电压Vgs<4v的场管（SI2302）,结果表现出了很好的驱动性能。



每个场效应管接一个大电阻下拉，目的是为了防止在单片机没接手电机的控制权时，电机由于PWM信号不稳定开始猛转。接一个下拉电阻，保证了场管输入信号要么是高，要么是低，没有不确定的第三种状态。那么电机也只有两种状态，要么转，要么不转。主控输出的是PWM波形，用于控制场效应管的关闭和导通，从而控制电机的转动速度。这就是crazepony电机驱动的原理。就是这么简单。

无刷电调（读做tiao）

在《电机与桨叶》一文中，我们提到大四轴基本上都是使用的无刷电机，无刷电机控制必须配合无刷电调使用。

无刷电调的输入是直流，通常直接接航模电池。输出是3根线，驱动无刷电机。另外无刷电调还有三根信号线，一般输入PWM信号，用于控制电机的转速。对于航模，尤其是四轴飞行器，由于其特殊性，需要专门的航模电调。



那么为什么在四轴飞行器上需要专门的电调呢，其有什么特别的地方？四轴飞行器有四个桨，两两相对呈十字交叉结构。在桨的转向上分正转和反转，这样可抵消单个桨叶旋转引起的自旋问题。每个桨的直径很小，四个桨转动时的离心力是分散的。不像直机的桨，只有一个能产生集中的离心力形成陀螺性质的惯性离心力，保持机身不容易很快的侧翻掉。所以通常航模直升机用到的电机控制信号更新频率很低，而航模四轴飞行器用到的控制信号更新频率很高。

四轴为了能够快速反应，以应对姿态变化引起的飘移，需要高反应速度的电调，常规PPM电调的更新速度只有50Hz左右，满足不了这种控制所需要的速度，且PPM电调MCU内置PID稳速控制，能对常规航模提供顺滑的转速变化特性，用在四轴上就不合适了，四轴需要的是快速反应的电机转速变化。用高速专用电调，IIC总线接口传送控制信号，可达到每秒几百上千次的电机转速变化，在四轴飞行时，姿态时刻能够保持稳定。即使受到外力突然冲击，依旧安然无恙。

对于四轴飞行器，我们一般使用无感无刷直流电机，电调利用第三相的感生电动势来测量转子位置。而对应的车模和船模，因为其电机需要频繁启动、停止、反转,而且对整套动力系统的重量也不是十分讲究,故用有感无刷电机，电调使用电机上的霍尔效应器件来检测转子的位置。

下图为车模/船模上使用的有感无刷直流电机及对应的电调，Hobbywing（好盈）公司的Quicrun系列3650型号。


QUICRUN
3650
SENSORED
PHASE
C B A


QUICRUN
SENSORED BRUSHLESS
10.5T





扩展阅读

- 《无感无刷直流电机之电调设计全攻略》——笔者最早看到这份文档，来自阿莫论坛。作者是timegate 墨鸢。对四轴飞行器无刷电机/电调入门和深入都非常合适，强烈推荐。在Crazepony的百度盘中提供一份下载。
- TI公司关于电机驱动和控制解决方案。

航模桨叶

> 作者：nieyong

桨叶

螺旋桨的型号由如8045，1038等4位数字表示，分别代表桨叶两个重要的参数，桨直径和桨螺距。桨直径是指桨转动所形成的圆的直径，对于双叶桨（两片桨叶，这是最常用的桨）恰好是两片桨叶长度之和，由前两位数字表示，如上面的80和10，单位为英寸。桨螺距则代表桨旋转一周前进的距离，由后两位数字表示，如上面的45和38。桨直径和螺距越大，桨能提供的拉（推）力越大。

以Phantom的桨叶为例。Phantom 1使用的是8045的桨叶，表示桨直径为8英寸（8×2.54=20.32厘米），桨螺距为4.5英寸；而Phantom 2使用的是9443的桨叶，表示桨直径为9.4英寸，桨螺距为4.3英寸。从桨叶的规格可以看出，Phantom的第二代能够提供更大的动力。

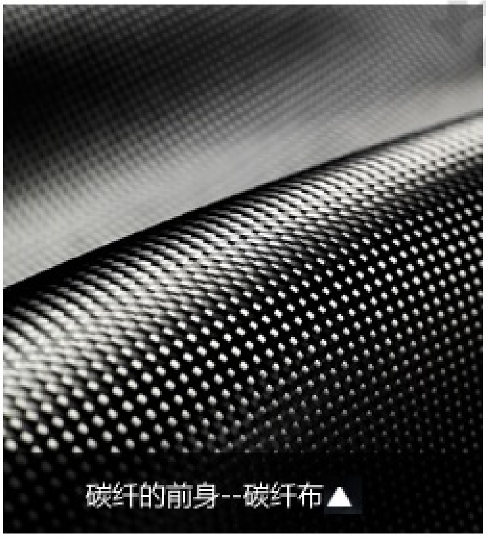
对于玩大四轴的人来说，桨叶的材质也是非常有考究的。根据桨叶使用材质的不同，现在市面上有的包括碳纤桨，木桨，注塑桨。

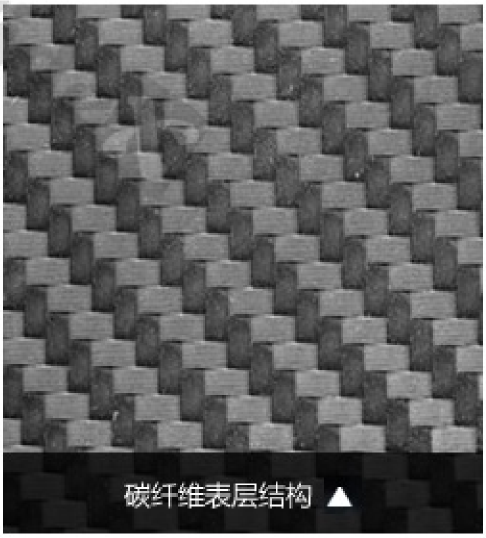
注塑桨：注塑桨是指使用塑料等复合材料制成的桨叶。在航模爱好者中以美国APC公司生产的桨叶最为有名，质量最好。淘宝上也有三种大类，分别是原装APC，ATG生产的假APC，以及不知名小厂生产的最廉价的有APC桨外观的塑料桨。

APC桨有一个衍生系列多轴专用的MR系列。优点是增强了桨的硬度，减少了重量，再次提高效率，解决了之前的高速转动桨尖变形问题。缺点是价格较贵，原装价格在89-128之间，并且无合适的垫圈，需要自己找物料缩小孔径。

碳纤桨

碳纤维是一种与人造丝，合成纤维一样的纤维状碳材料。由于碳纤维的材料原因，它有优异的硬度和合适的桨型，非常适合技巧性飞行和3D飞行。碳纤桨效率优于木桨，价格比木桨更贵，稍微低于原装APC。





几种不同的碳纤桨叶。



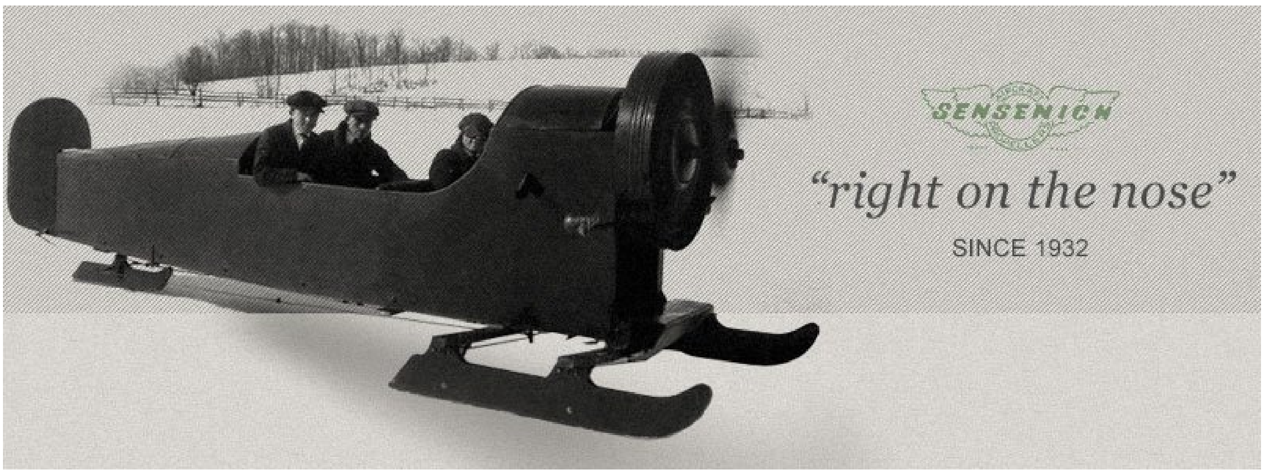






木桨

木桨的材料多为榉木，硬度高，重量轻，经过风干打蜡上漆以后不怕受潮。在航空史上，木桨在早期扮演了非常重要的角色。一战时期的很多飞机都是使用的木桨，后来才逐渐被铁桨取代。专门提供飞行器的木桨的公司sensenich。



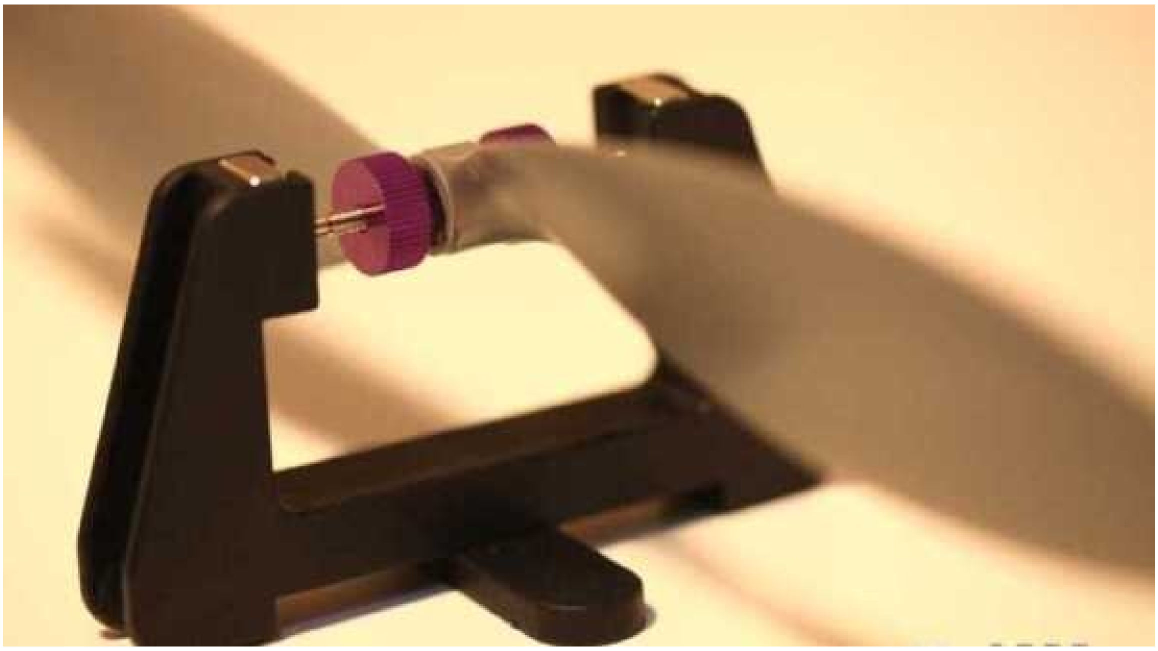
在航模爱好者中，木桨主要用在固定翼飞行器上，最为高端的是欧洲XOAR公司的木桨。



下面是一个提供了很多关于螺旋桨知识，历史，设计的网站heliciel。

桨叶的平衡

桨叶会随着电机高速旋转，所以其旋转平衡是一个很重要的指标。如果桨叶有一头重一头轻，或者一侧轻一侧重等不平衡问题，那么在旋转时就会产生左右/前后晃动等问题。这样不仅仅动力效率低，而且还可能会损坏电机造成航模飞机炸机。一般的桨叶都已经进行过自平衡测试，对于要求高的场合也可以自己调节桨叶的自平衡。下面是一种常见的测试自平衡的工具。



当然，对于像Crazepony这样的小四轴，一般就很少考虑桨叶自平衡问题。

桨叶和电机的组合

不同的桨叶和电机（KV值不一样）能够形成不同的动力组合，适合于不同的航模飞机和应用场景。

无刷电机绕线匝数多的，KV值低，最高输出电流小，但扭力大。达到同样的推力，要比高KV值的剩电，所以四轴飞行器多使用小KV的电机。

无刷电机绕线匝数少的，KV值高，最高输出电流大，但扭力小。同样的设备重量（电机、电调、电池），得到的最大推力要高过低KV值的电机。

简单的说，相同的电机和电池，大KV值用小的螺旋桨，小KV值用大的螺旋桨。相对来说螺旋桨配得过小，不能发挥最大推力。相对来说螺旋桨配得过大，电机会过热，会使电机退磁，造成电机性能的永久下降。

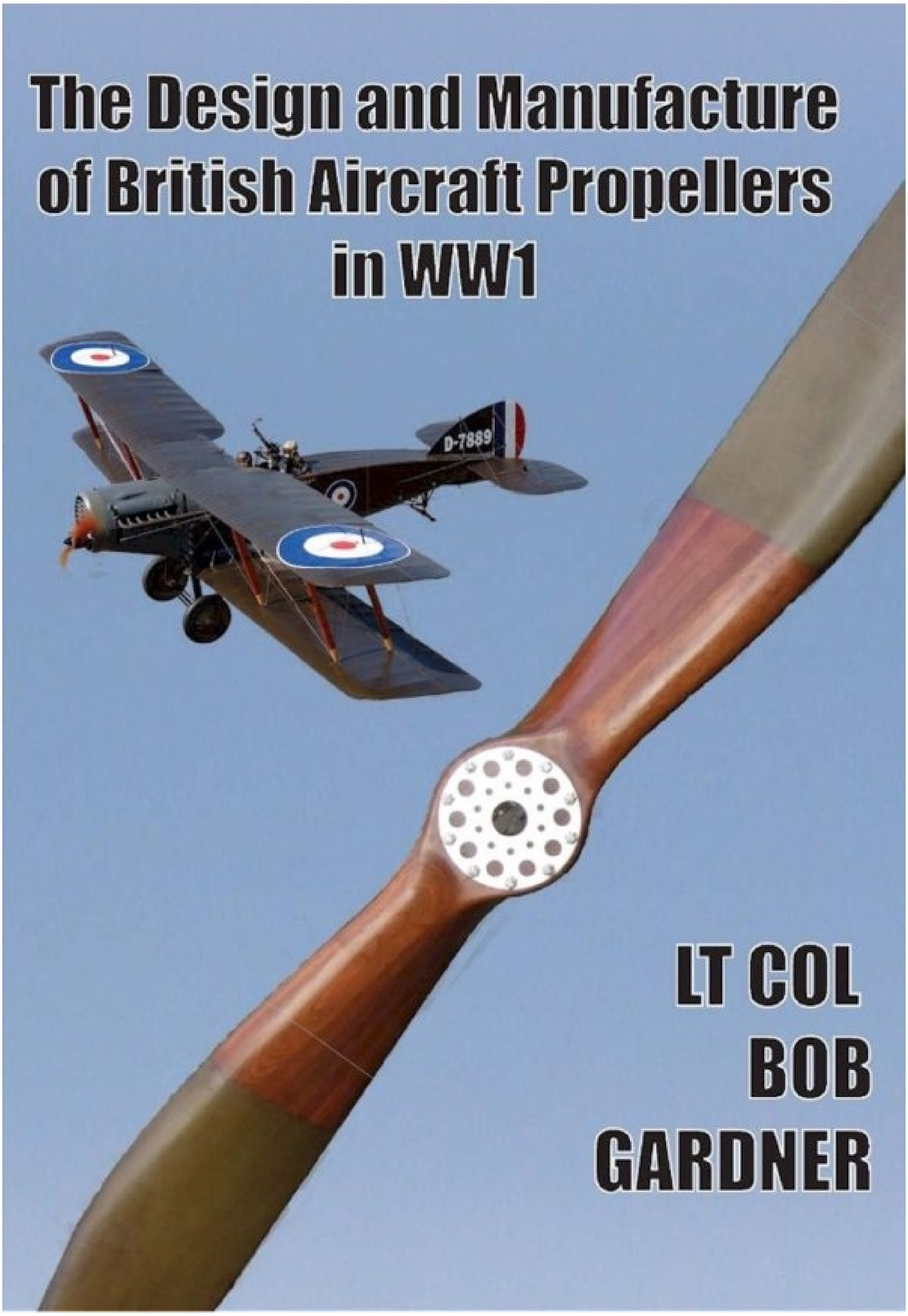
原则上，更小的KV值和更大的桨叶，能够表现出更好的动力效率。也就是说相同的电池，能够飞行的时间更长。例如X5C这样的玩具飞机，电机和桨叶由减速齿轮连接。减速齿轮降低了电机的KV值，有更好的动力效率。Crazepony现在使用30000转/分钟的空心杯电机，75mm桨叶。当配备350mAh电池时，能够飞行4-5min左右。在电机和桨叶的动力效率组合方面，应该还有进步的空间。

扩展-木桨的故事

木桨在人类航空早期扮演了非常重要的角色。一战时期的很多飞机都是使用的木桨，后来才逐渐被铁桨取代。下面是那个时候木桨的一些珍贵图片。







更多关于木桨的知识，参考Wooden Propellers(A site dedicated to the preservation of early wooden airplane propellers)。

飞行器控制器——主控MCU

> 作者：nieyong

四轴的灵魂——飞控

飞控是四轴的灵魂。那么什么是飞控呢？飞控其实就是四轴飞行器的电子控制部分，包括传感器部分惯性导航模块和控制部分的MCU。

四轴飞行器相对于常规航模来说，最复杂的就是电子部分。可以和其它两种常见的航模固定翼以及直升机比较一下。在常规固定翼航模上，陀螺仪并非常用器件。相对操控难度大点的直升级航模，如果不做自动稳定系统，也只是锁尾才用到陀螺仪。四轴飞行器则必须配备陀螺仪，这是最基本要求，不然无法飞行，更谈不上飞稳了。不但要有，还得是3轴(X、Y、Z)都得有，这是四轴飞行器的机械结构、动力组成特性决定的。在此基础上再辅以3轴加速度传感器，这6个自由度，就组成了飞行姿态稳定的基本部分，也是关键核心部分惯性导航模块，简称IMU。飞行中的姿态感测全靠这个IMU了，可见它是整架模型的核心部件。

IMU感知飞行器在空中的姿态，将数据送给处理器MCU。处理器MCU将根据用户操作的指令，以及IMU数据，通过飞行算法控制飞行器的稳定运行。由于有大量的数据需要计算，而且需要实时性极高的控制，所以MCU的性能也决定了飞行器是否能够飞得足够稳定，灵活。

本篇文章只讲飞控的MCU部分，将有另外一篇文章《飞控——传感器》讲解IMU部分内容。

主控MCU STM32

Crazepony的主控MCU选用的是意法半导体的STM32f103T8U6，为32位ARM Cortex-M内核，最高72MHz。关于为什么会选用这片MCU，在CamelGo的博客我和Crazepony的那点事儿(2)中有这么一段描述。



> 曾经在犹豫用TI的430系列单片机还是意法半导的STM32。那是在我大二的时候，从来没有接触过STM32，以前都是用51单片机和TI的msp430系列单片机。好吧，我承认了，我做Crazepony其实是就是为了学习STM32的，没有买过STM32相关的开发板，就这么简单粗暴大刀阔斧的开始了我的STM32之旅…
>
> 最终选择用STM32当然还有其他原因，TI公司的MSP430系列都是基于低功耗在做文章，作为移动消费电子，对电源续航能力要求比较高的场合比较适用。
>
> ……
>
> 之所以选择STM32F103T8U6作为Crazepony的主控芯片。首先因为他是crotex-M3内核，继承了ARM的优良性能，主频能跑到72MHz，3个通用定时器，1个高级定时器，7通道DMA控制器，而且总线接口资源丰富；其次是因为它VFQFPN36的封装，只有6mm*6mm的占地面积,对这个寸土寸金的项目来说简直太赞了。这么高的性价比，当然让我选择了他作为主控。72MHz虽然生不了孩纸，但是足以处理除了图像之外的大部分任务了。

主控选型需要考虑的问题

CamelGo说是误打误撞就选用了STM32作为主控MCU，其实真正在选型时候，有很多问题是需要我们考虑的，下面列举出几个最常见的问题。

- 首先是MCU的性能，最重要的指标就是主频。这直接决定MCU计算的快慢。四轴飞行器有很多来自IMU的数据需要处理，而且还有复杂的控制算法，如果MCU的性能不够，那么将直接限制飞控只能够处于一个比较初级的阶段，无法完成更加复杂的功能和精准的控制；
- MCU的接口也很重要。MCU是整个四轴飞行器的大脑，几乎所有的数据都要连接到它上面。例如I2C总线个数，DMA通道数目，GPIO数据等等；
- 正如博客中提到的，MCU的尺寸大小也是值得考虑的因素；
- 另外就是这个MCU的开发是否简单，技术资源的支持是否足够多。这对于一个DIY的开源四周飞行器也很重要；

综合了这些因素，我们选择了STM32这片MCU作为我们的主控MCU。我们也高兴的看到，很多国内国外的四轴爱好者也选择了这一片MCU。Crazepony不再孤独！

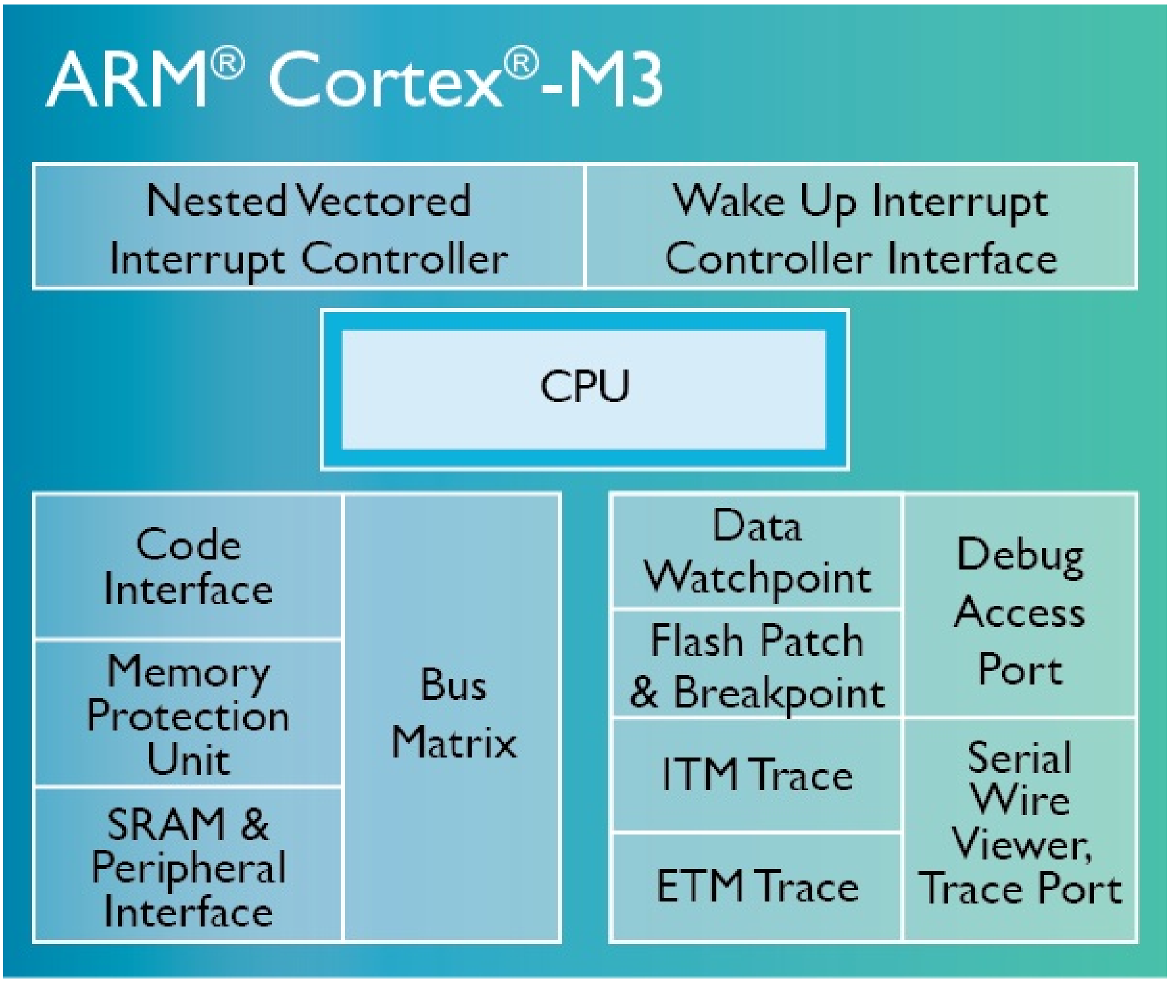
关于ARM Cortex-M

由于STM32使用的是ARM Cortex-M架构，所以这里有必要做一点ARM Cortex的普及。

ARM公司在经典处理器ARM11架构之后，为了给不同需求的CPU厂商提供服务，之后的内核架构命名都改为Cortex，并分成了A,R,M三类,也即将ARM的三个字母拆分为三个架构的名，代表着不同的发展方向：

- A系列处理器可托管丰富的OS平台和位应用商提供全方位的解决方案，诸如低成本手机、数字电视、机顶盒、打印机、服务器等。
- R系列为实时处理器，要求可靠性、可用性、可维护性和实时响应的嵌入式系统提供解决方案。
- M系列是一系列可向上兼容的高效能、易于使用的处理器，这些处理器旨在帮助开发者满足将来嵌入式的需要，这些需要包括低成本、不断增加的连接、代码改善移植等。M系列主要应用在智能测量、人机接口设备、汽车电子、工业控制、大型家电等。

所以，我们的STM32使用的Cortex-M3内核，就属于ARM的M系列，主要针对嵌入式产品需求而设计的。下面是一张Cortex-M3的内核架构图。



飞行控制器——传感器

> 作者：nieyong

在《飞行控制器——主控MCU》一文的四轴的灵魂——飞控章节中已经提到，飞控是由传感器部分（即惯性导航模块，IMU）和控制部分MCU组成。这篇文章我们将重点讲讲传感器部分的知识。

四轴飞行器的传感器就是用来感知飞行器在空中的姿态和运动状态，这有个专有名词叫做运动感测追踪，英文Motion Tracking。在消费电子市场，运动感测技术主要有四种基础运动传感器，下面分别说明其进行运动感测追踪的原理。

加速器(G-sensors)

加速器可用来感测线性加速度与倾斜角度，单一或多轴加速器可感应结合线性与重力加速度的幅度与方向。含加速器的产品，可提供有限的运动感测功能。例如，含加速器的产品，可在固定方位，感应縱向至横向的移动，因此，加速器主要使用于与重力方向有关的感测产品中，可提供如将手机的縱向萤幕转换为横向等的简单功能。

陀螺仪(Gyros)

陀螺仪是利用高速回转体的动量矩敏感壳体相对惯性空间绕正交于自转轴的一个或二个轴的角运动检测装置。利用其他原理制成的角运动检测装置起同样功能的也称陀螺仪。

陀螺仪可感测一轴或多轴的旋转角速度，可精准感测自由空间中的复杂移动动作，因此，陀螺仪成为追踪物体移动方位与旋转动作的必要运动传感器。不像加速器与电子罗盘，陀螺仪不须借助任何如重力或磁场等的外在力量，能够自主性的发挥其功能。

更加详细的资料，参考MPU6050使用相关的篇章。

电子罗盘(E-Compasses)

电子罗盘也叫数字指南针，磁力计，是利用地磁场来定北极的一种方法。古代称为罗经，现代利用先进加工工艺生产的磁阻传感器为罗盘的数字化提供了有力的帮助。现在一般有用磁阻传感器和磁通门加工而成的电子罗盘。

电子罗盘可藉由地球的磁场来感测前进方向。运用电子罗盘的消费性电子产品应用，包含在手机的地图应用程序显示正确方向，或为导航应用程序提供前进方向数据。然而，电子设备或建筑材料的磁场干扰，比地球磁场来得强，导致电子罗盘传感器的输出值，较容易受到各种环境因素的影响，尤其在室内更是如此，因此，电子罗盘须要透过频繁的校正，才能维持前进方向数据的准确度。

压力传感器(Barometers)

压力传感器又叫做气压计，会藉由气压的变化来感测物体的相对与绝对高度，常被运用于与运动、健身、方位推测等应用有关的消费性产品中，例如，可感测使用者的移动层楼，调整地图信息。

微机电系统(MEMS)

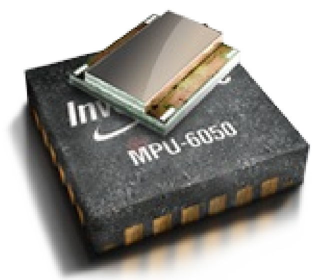
微机电系统（MEMS, Micro-Electro-Mechanical System）是一种先进的制造技术平台。微机电系统是微米大小的机械系统，是以半导体制造技术为基础发展起来的。

在我们的四轴飞行器上用到的是传感MEMS分支。传感MEMS技术是指用微电子微机械加工出来的、用敏感元件如电容、压电、压阻、热电耦、谐振、隧道电流等来感受转换电信号的器件和系统。

我们使用的加速度陀螺仪MPU6050，电子罗盘HMC5883L都是微机电系统。

crazepony传感器

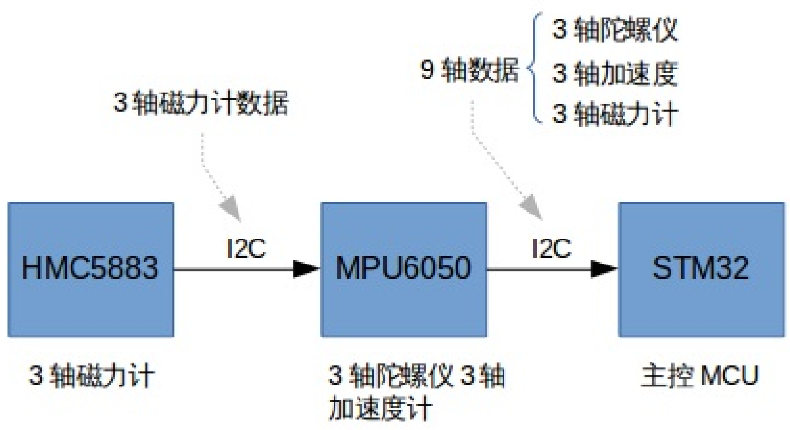
Crazepony使用传感器MPU6050，集成了3轴陀螺仪和3轴加速度传感器。



关于传感器的选型，在CamelGo的博客我和Crazepony的那点事儿(2)中有这么一段描述。

> Crazepony采用的是最常用的MPU6050陀螺仪加速度计一体芯片，成本不超过20元，对小四轴来说，它的精度和性能绰绰有余了（当我听说教研室师兄用的一颗传感器裸片卖1W+时，我整个人都不好了..），MPU6050在这个价位里面几乎是占有绝对的性价比优势。首先，它将陀螺仪和加速计整合在一个片上，通过IIC总线给出六个维度的ADC值；其次，芯片本身提供一个"从"IIC接口，供用户接第三方的IIC器件，一般选择是接一个电子罗盘，如HMC5883L,构成一个9轴的输出的姿态模组,现在MPU9150已经丧心病狂的把电子罗盘功能也整合在片上了,但是要买60+元；最后，这颗芯片内部集成了一个DMP（Digital Motion Processor）处理器，这是最让我爱不释手夜不能寐的一个功能,直接硬件解算四元数,从某种程度上说解放了20%的主控资源

在crazepony上，MPU6050,HMC5883传感器之间的连接如下图所示。



航模遥控器

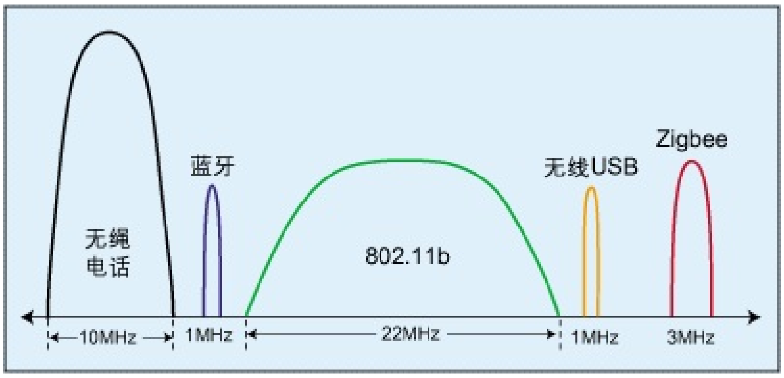
> 作者：nieyong

常见的无线遥控按照使用技术不同，主要有红外遥控和**2.4G**遥控。红外遥控有方向范围窄，距离短等特点，所以航模遥控器基本上都是使用2.4G无线遥控。 我们知道蓝牙，wifi，ZigBee等都是采用的2.4G频段，那么这个又有什么差别呢？首先我们需要对2.4GHz无线技术以及标准做一个简单的介绍。

2.4GHz无线技术简介及其标准

2.4GHz无线技术如今已经成为了无线产品的主流传输技术。所谓的2.4GHz所指的是一个工作频段2400M-2483M范围，这个频段是全世界免申请使用。常见的Wifi， 蓝牙，ZigBee都是使用的2.4G频率段。

Wifi， 蓝牙，Zigbee都是基于2.4GHz的，只不过他们采用的协议不同， 导致其传输速率不同，所以运用的范围就不同。同样是采用2.4G频率作为载波，但不同的通讯协议衍生出的通讯方式会有着天壤之别；仅仅在传输数据量上，就有着从1M每秒到100M每秒的差别。详见下面的2.4GHz ISM频段无线信号比较图。



2.4G无线通信模块

2.4G无线通信模块（英文：2.4Ghz RF transceiver / receiver module）工作在全球免申请ISM频道，频率范围是2400MHz-2483MHz。实现开机自动扫频功能，共有50个工作信道。市面上的航模遥控器基本上都是使用2.4G无线通信模块，对应遥控器会配置一个接收机。遥控器（发射机）和接收机需要配对使用，不同厂商的不能够兼容，因为使用的收发芯片以及协议可能不一样。

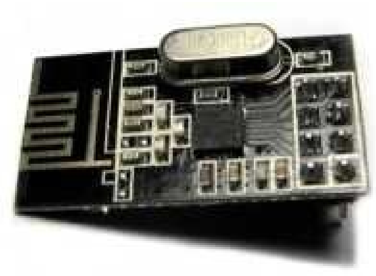
Crazepony使用的**2.4G**无线射频芯片**NRF24L01**作为无线控制。在飞机上我们将NRF24L01作为接收部分做到PCB板上，在遥控器上我们使用基于NRF24L01芯片的模块，方便插拔跟换。

crazepony的2.4G模块

Crazepony使用的是单片2.4G无线射频收发芯片NRF24L01。关于为什么会选用这片IC，在CamelGo的博客我和Crazepony的那点事儿(2)中有这么一段描述。

> 数据通信这块，也没什么特别要考虑的，短距离高速通信且免费这一点要求就限制了只能选择2.4GHz这个频段，在这个频段出了很多优秀的芯片厂商。在学生时代，我用得最多是Nordic公司的NRF24L01这个系列的收发一体芯片，由于刚开始着手启动Crazepony这个项目时，我只会这颗操作芯片，本着方便的原则，所以很自然的选择了这颗它（后来发现国外一个团队bitcraze也用的这个系列的芯片时，还是有点小激动的）

市面上有基于NRF24L01芯片的模块，有统一的标准接口。在遥控器上，我们就直接使用了这个模块，用来和飞控上的NRF24L01进行通信。我们标配的模块如下图所示，控制的距离在20m左右。



遥控器上也可以使用带有PA（功率放大，power amplifier）的NRF24L01模块，带有鞭状天线，我们实测能够达到100m的控制距离。



下图为使用带有PA的2.4G模块遥控器的实物图。



市场上航模遥控器介绍

进口遥控，有名的品牌有JR/FUTABA/SANWA等。

- 油动版（标准接收机），遥控距离在600~800米（4VF/4EX/6EX/MAX66/RD6000/VG400），一般使用FM/PPM模式
- 电动版（微型接受机），遥控距离在400~500米（一次变频版）700~1000米（二次变频）
- 高级遥控器，象FF9/JR9X2/JR12/T12/T14，PCM版二次变频接收机，最远有1.5KM~2KM



国产遥控器，最常见的为天地飞，英文WFLY。

- 天地飞6通，一次变频版接收机一般遥控距离在300~400米，这种小型接收机一般只适用于电动飞机，装到油机上面有一定危险。
- 天地飞8，9通，遥控距离600~800米，二次变频版，可以飞油动飞机，电动飞机，但飞汽油机有危险。



国产遥控与进口遥控的差距区别在于，一是无线电技术上的差距，另一方面国产的研发机构太少，没有经过长时间的产品测试。国产遥控器用的都是比较便宜的电子元件，所以成本比较低，特别是接收机，接收机的芯片，好接收机都是进口芯片，所以很稳定，程序编写比较好，便宜的接收机芯片比较便宜，所以容易出事。一个很简单的例子，遥控器上面的电位器，国产设备用得是国产电位器一般寿命在10~30万次，进口设备的电位器，像FUTABA是台湾产的可以达到100万次，这样直接影响遥控器的寿命。一般认为遥控器的做工方面JR遥控器最好，线路布局各方面都很好，特别是目前只有JR遥控器配有保险丝如果爱好者不小心短路了，只需要更换保险丝就可以，比较人性化，更重要的是JR遥控器全部是在日本，马来西亚制造，品质有所保证。 FUTABA大众型遥控器现在比较普遍，现在FUTABA的低端遥控器都是在国内产的，接收机是台湾产的，所以遥控器这东西一份价格一分货。模友按照自己的经济能力去选购就可以了，千万不能攀比，选择适合自己的都可以。

现在国产遥控器的厂家越来越多，价格的优惠使得航模的门槛越来越低，所以有一款适合自己的遥控器是广大模友的心愿，广大模友按照自己的经济实力合理 选购属于自己的遥控器。

拥有一个遥控器，能否用来控制Crazepony四轴飞行器

要使用原有的遥控器控制Crazepony四轴飞行器，需要满足下面两个条件。

- 遥控器是使用和crazepony飞机上2.4G射频接收芯片NRF24L01一样的发射芯片，或者相兼容的芯片。
- 需要知道遥控器的数据编码格式。很多厂商原则上是不会公开遥控器使用的数据编码格式的。这就需要对该厂商遥控器的编码进行破解。

例如crazyflie就支持ESky的遥控器，因为该遥控器的编码被大神破解并且公开。所以有人专门争对这个提供了接收机代码包，叫做Esky Protocol，其实就是ESky公司遥控器使用的通信协议。只要在飞控端将遥控器的数据解析出来，就可以使用ESky航模公司的遥控器进行控制。



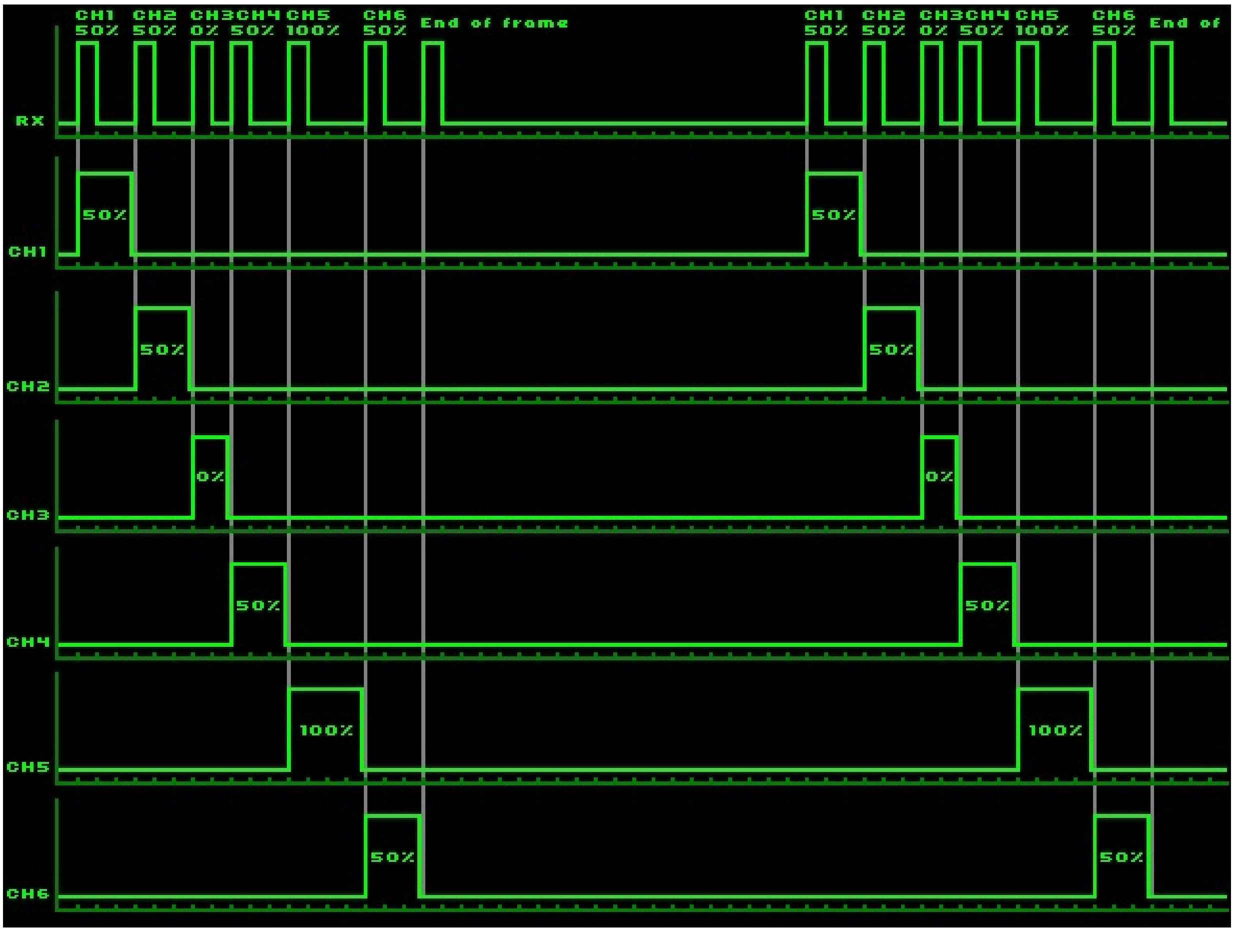
关于ESky公司的2.4G遥控器设备的通信协议，参考ArduinoRCLib项目中的描述。

> The Esky 2.4 GHz equipment uses the Nordic NRF2401AG in both the transmitter and receiver. A compatible alternative to this chip is the NRF24L01+ (which is widely available).

关于接收机的输出和PPM/PCM编码

todo

关键字：Arduino PPM



关于遥控曲线问题

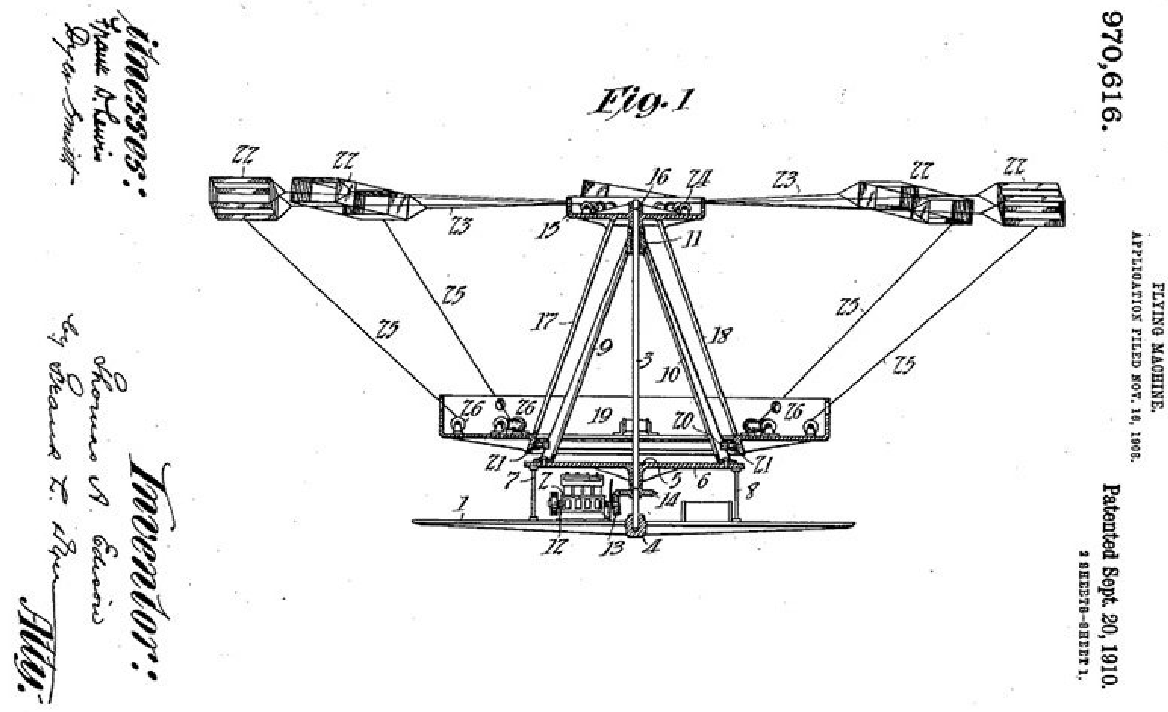
很多新手反应，航模不好控制。其实这是有原因的，其中最大的一点就是关于遥控曲线的问题，而其中又以油门曲线最为有代表性。

目的是把直线变化的油门，变为曲线变化，以此提供不同的飞行模式。我们以最简单的3点曲线来说明，我们把发射机油门遥杆从下底端，中段，上顶端分为3个点，普通的发射机对应的油门量分别是0%，50%，100%，如果具有油门曲线的发射机，则可对这3个点单独进行设定。

比如，我们将下底端的0%设定为100%。这时，油门摇杆的位置在中段时油门量为50%，向上向下推动油门遥杆都是不断的增加油门量直到100%油门。这时我们看到的是一个V字形变化的油门曲线了(这是3D模式的油门变化要求)。5点曲线就是在3点之间插入2个点，以提供更接近曲线的平滑设定。当然还有一些高端的遥控器提供了7点甚至更多的设定点。那么多少合适呢，对于世界级的比赛其实5点或以上就已经足够了！

Crazepony原理讲解

本章主要讲解Crazepony的硬件原理和软件框架。从中可以看到我们的主要设计思路。同时，我们将元器件选型和开发经验进行了总结，方便大家DIY属于自己的四轴飞行器，并且少走我们曾经走过的弯路。



Crazepony器件总览及选型说明

> 作者：nieyong

我们暂时不提供桨叶/空心杯电机/电池等易损耗配件的购买。Craze团队整理了部分淘宝上的购买地址及注意事项，大家可以自行购买。

电机 & 桨叶

720有刷空心杯电机，中心轴直径1mm。淘宝购买地址：地址1。



直径为75mm桨叶，桨叶中心孔直径为1mm，正好配中心轴直径为1mm的720电机。淘宝购买地址：地址1。



电池

Crazepony飞机电池为650mAh，3.7V，25C航模电池。大家也可以选择350mAh电池，购买地址（建议自己上淘宝上寻找，这个链接是阿里巴巴链接，需要批量拿货）。

购买时需要注意电池接口。

2.4G无线射频

Crazepony机身上使用2.4G无线射频收发芯片nRF24L01。



遥控器上使用2.4G无线射频模块，根据遥控距离的不同，有下面两种可以选择。

加功率放大（PA）的鞭状天线，最大发送功率20dBm，可以遥控距离100米。购买地址。

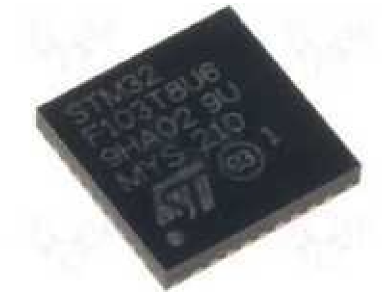


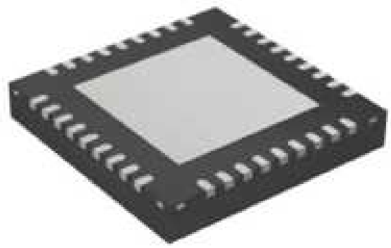
不加PA的发射模块，最大发射功率0dBm，可以遥控距离20米。购买地址。



主控MCU（单片机）

主控为STM32的F10系列，STM32F103T8U6，ARM Cotex-M3内核，72MHz主频，128KB存储空间，BGA封装。





电机驱动

SI2302，N沟道增强型MOS，导通电阻82mΩ，2.2v低GS电压。

陀螺仪加速度计

MPU6050，三轴陀螺仪+三轴加速度计，自带DMP四元数输出，内部温度补偿。



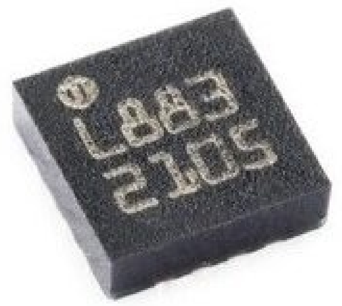
气压计

MS5611，10cm精度气压计。



电子罗盘（磁力计）

HMC5883L，电子罗盘。



蓝牙透传模块

HM-06

淘宝购买地址，仅此一家。

USB转串口芯片

cp2102，USB转串口芯片，用于固件下载，参数调试。

锂电池充电管理

LT4054，锂电充电管理芯片。

摇杆电位器

淘宝购买地址。

我们对油门摇杆的要求是推左右自动回中，推上下不回，可停留在另意位置。根据这个要求，我们只找到了这种拆机的摇杆物料。

5.1版本即以后，油门摇杆使用自动回中的电位器，不再使用上面的拆机摇杆，所有摇杆都是新物料。

陶瓷天线

飞机上2.4G使用了陶瓷天线，型号为：rainsun An9520-245。

Crazepony硬件原理讲解

> 作者：CamelGo

原理图设计

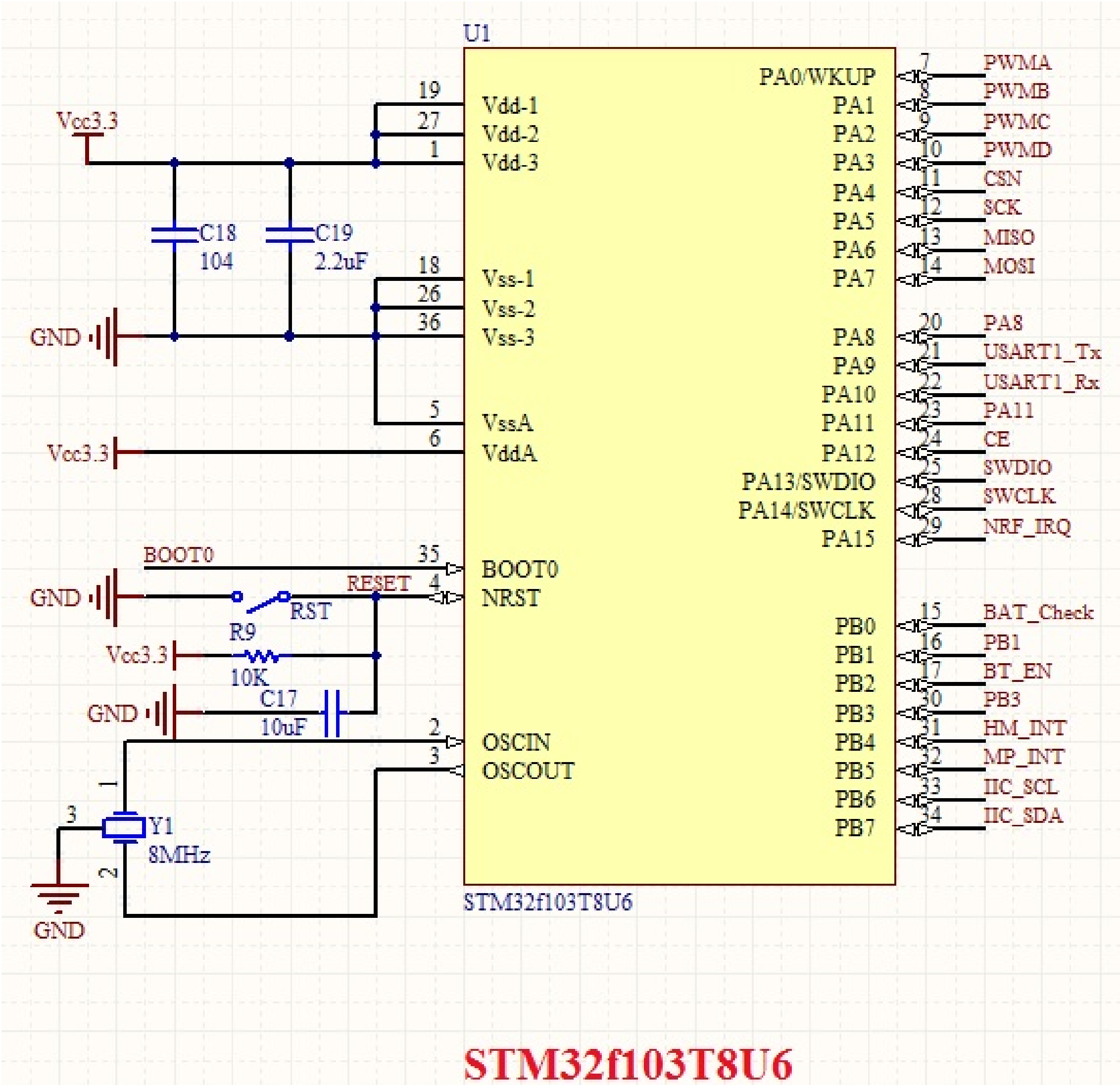
Crazepony-II(第4版/4.1版)硬件组成

- 主控：STM32f103T8U6
- 姿态传感器：MPU6050（3加速度+3角速度）
- 电子罗盘 ：HMC5883L
- 无线通信协议：NRF24L01+ 与 蓝牙2.1/蓝牙4.0透传共存
- 有线通信协议：CP2102（USB转串口）
- 电机驱动：SI2302场效应管
- 外部接口:标准mircoUSB接口
- 电机：Coreless高速电机 3W转/分钟
- 桨叶：46mm黑色正反桨
- 电池：350mAh 25c航模动力电池

Crazepony-II(第4版/4.1版)原理图设计简介

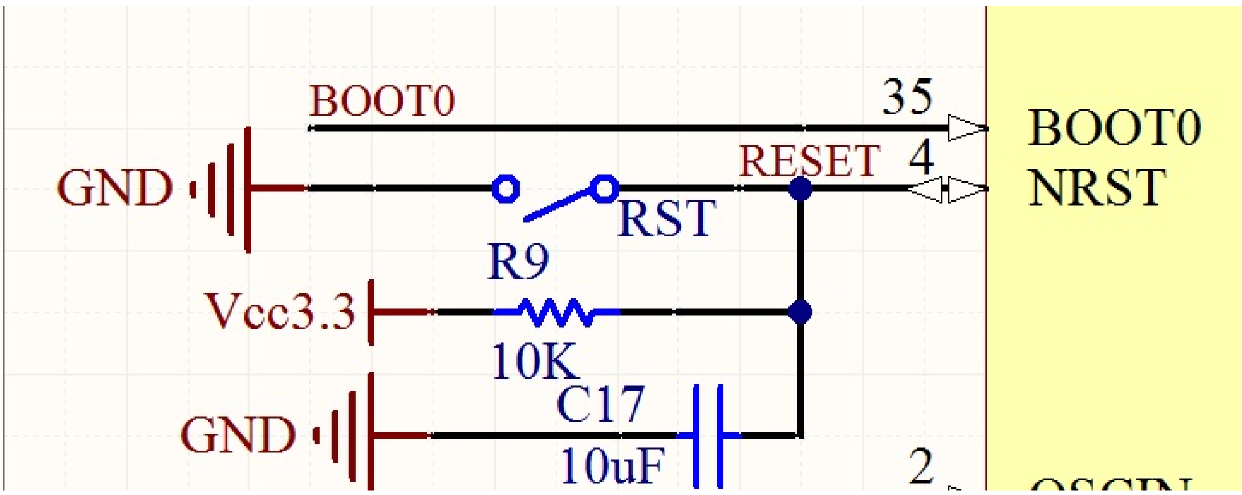
单片机最小系统

单片机在任何一个系统里面，无论他扮演什么角色。它要正常工作，都必须要有一个最小系统。下面，从最小系统开始简介Crazepony：



STM32的最小系统一般包括：复位电路，外部时钟电路，启动模式选择电路，电源退偶电路等

复位



查阅意法半导的官方手册可以知道，STM32系列单片机都是低电平复位。于是采用如图的主流复位电路作为Crazepony主控的复位电路

外部时钟

官方手册有写，外部时钟我们采用的是8M无源晶振。单片机内部做倍频，系统时钟最高可达到72M

启动模式

STM32的启动模式分为三种，可以下面的表格给出：

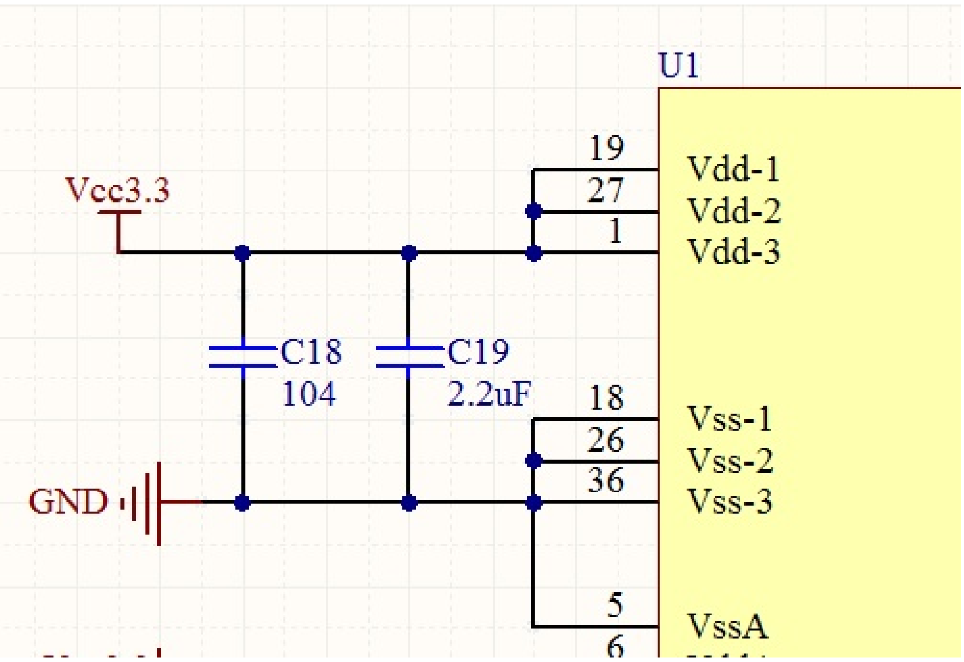
BOOT1	BOOT0	启动模式	描述
X	0	从用户闪存启动	正常工作模式
0	1	从系统存储器启动	厂家设置用
1	1	从内置 SRAM 启动	这种模式用于调试

由于为了方便用户调试和二次开发，所以Crazepony采用了SWD在线程序调试接口和ISP程序下载两种方式。SWD调试接口可以使用编译调试器在线对程序进行仿真、调试、下载，这对开发人员来说是很方便的，缺点就是需要PC端有一个这样的软件来支撑。串口ISP下载方式，只需要STM32的UART1的两个数据 线，就能将编译生成的*.HEX文件烧写进单片机，不足之处是不能仿真程序。

关于启动模式的应用，在USB-Serial处会有介绍。

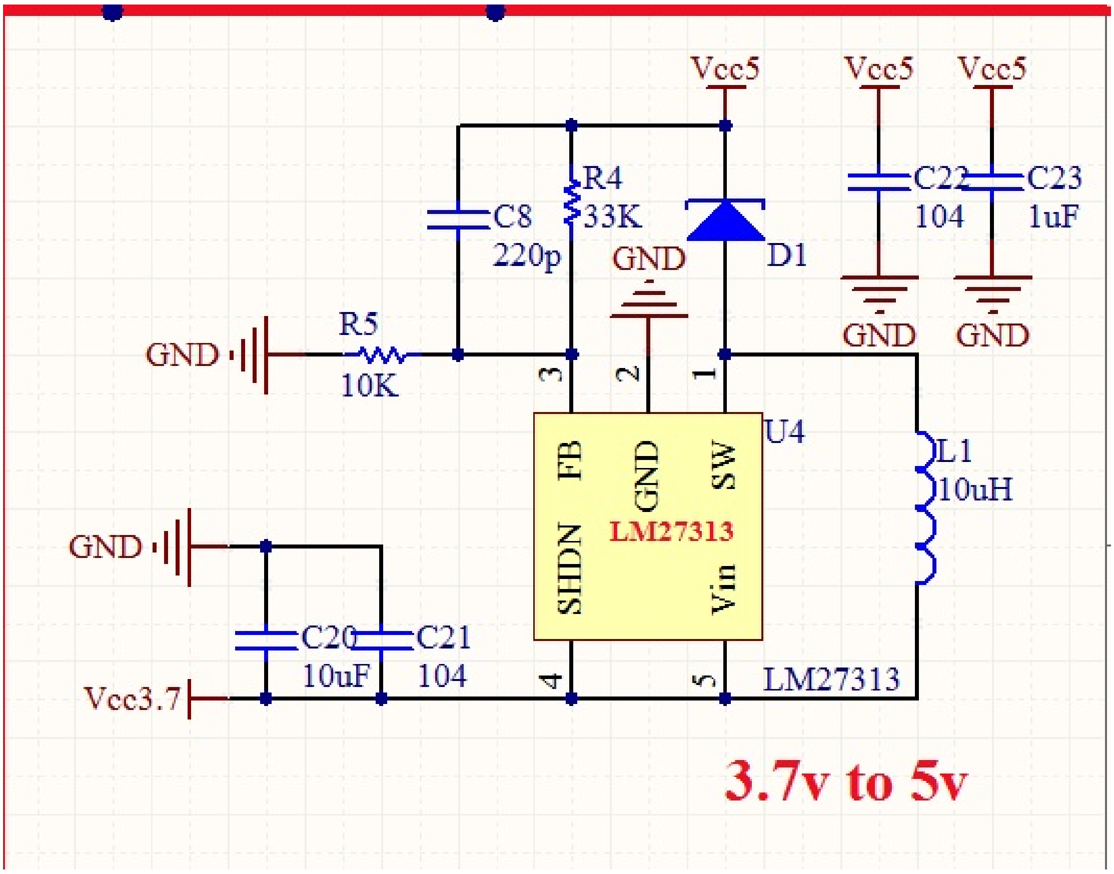
电源退偶

不仅是主控最小系统需要对电源退偶，所有的数字电路和模拟电路共存的系统，都需要对电源退偶。电源退偶，说直接一点就是将电源上的噪声电压引入到地平面，让电源电压保持在一个稳定的值，这样系统才可能稳定工作。怎么做呢？用一个大电容并联一个小电容。



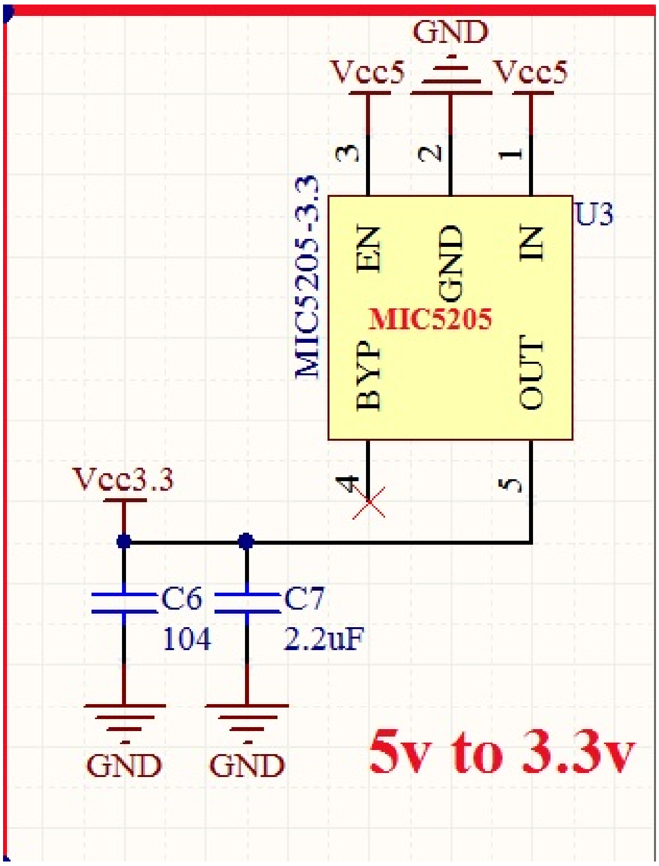
我们都知道，电容对频率越高的信号，呈现低阻特性，对直流呈现高阻特性。那么电源上的噪声对地平面而言，就是一个交流信号，交流信号就能通过电容到达地平面，而电源是一个直流，电容对他呈现出无限大的阻力，无法通过。这样，我们用示波器就可以看到，加了退偶电容的电源会比没加退偶电容的电源，波形要稳定得多。

系统电源

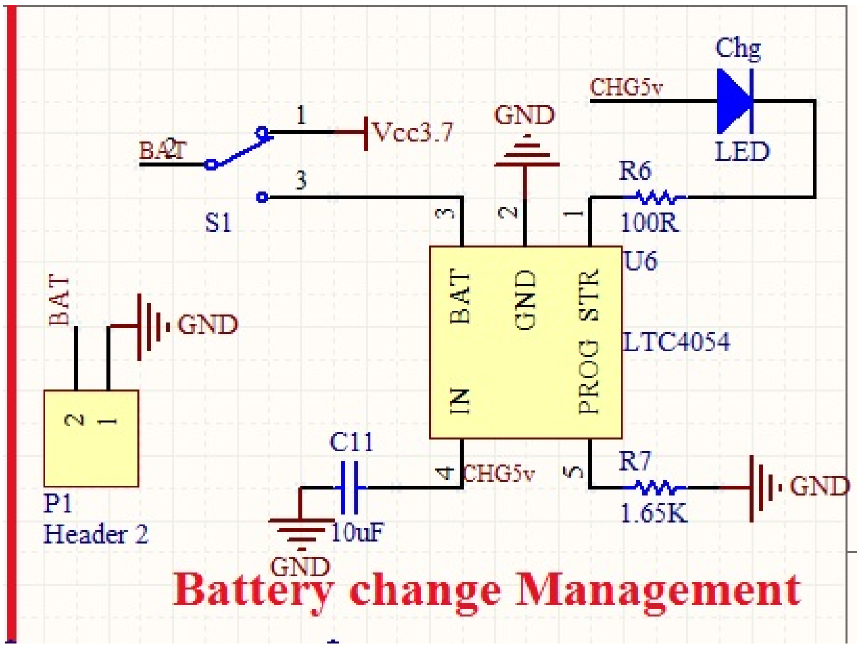


市面上所有的航模动力电池，都是3.7V的标称值，比此电压高的电池，都是几个3.7V的电池串联起来的

Crazepony采用了一节动力电池，电池电压是3.7V，而系统所有芯片都要求是3.3v供电。3.7V到3.3V只有0.4v的压差，我们考虑过采用低压差的LDO稳压芯片输出，但是要知道，四个空心杯电机转起来以后，瞬间电流能达到3A，此时电池电压会被拉低到一个LDO无法正常工作的值，于是我们后来放弃了直接将电池接到LDO稳压芯片上，而是在中间采用一个过渡的电路：一个DC-DC的升压电路，首先将电池电源升到5V左右，再接入LDO芯片（MIC5205-3.3）如图



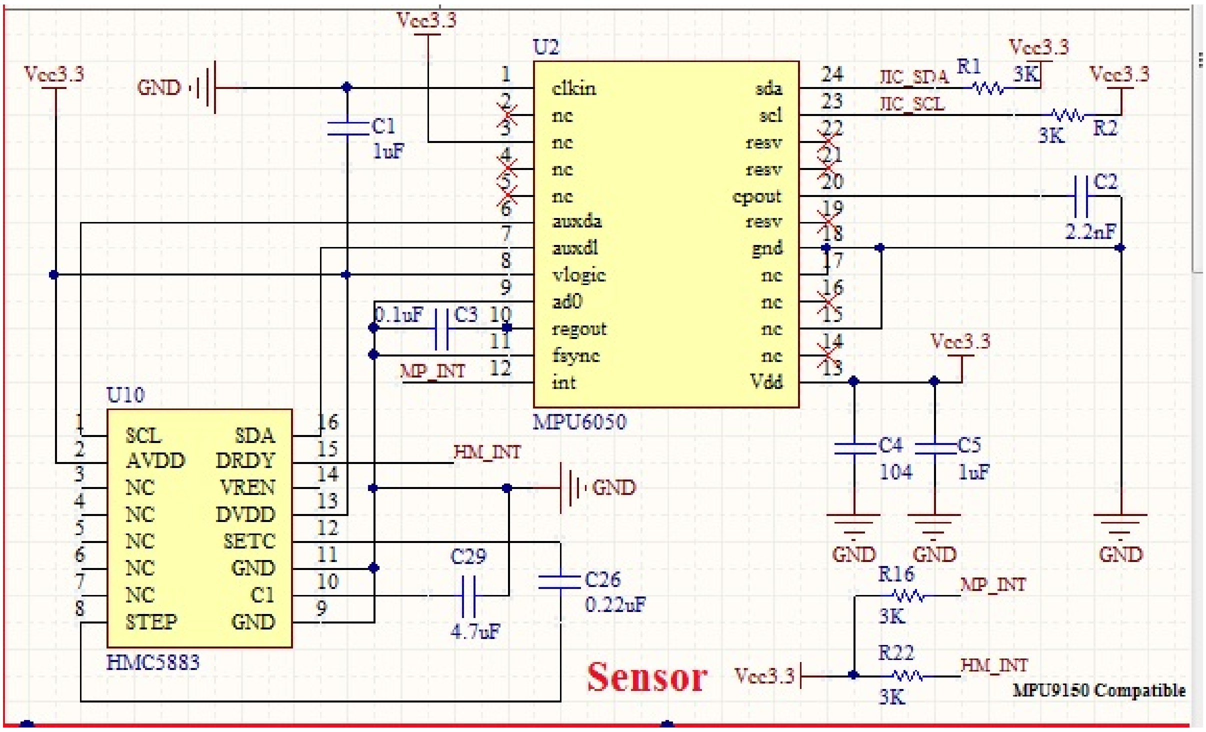
锂电池充电管理



锂电池充电这一块，采用的是 LTC4054，外部电路简单，一个电阻R7作为充电限流电阻，充电电流最大可达600mA，充电电流计算公式： `IBAT =(VPROG /RPROG)*1000` 。

R6作为充电指示灯的限流电阻，选择几百欧姆就行了。当充电进行中，引脚STR常低，充电结束时，STR拉高。对应的状态就是：充电时，CHG灯常亮，充电完成，CHG 灯灭。

姿态传感器



Crazepony采用的是最常用的MPU6050陀螺仪加速度计一体芯片，成本不超过20元，对小四轴来说，它的精度和性能绰绰有余了（当我听说教研室师兄用的一颗传感器裸片卖1W+时，我整个人都不好了..），MPU6050在这个价位里面几乎是占有绝对的性价比优势。首先，它将陀螺仪和加速计整合在一个片上，通过IIC总线给出六个维度的ADC值；其次，芯片本身提供一个“从”IIC接口，供用户接第三方的IIC器件，一般选择是接一个电子罗盘，如HMC5883L，构成一个9轴的输出的姿态模组，现在MPU9150已经丧心病狂的把电子罗盘功能也整合在片上了，但是要买60+元；最后，这颗芯片内部集成了一个DMP（Digital Motion Processor）处理器，这是最让我爱不释手夜不能寐的一个功能，直接硬件解算四元数，从某种程度上说解放了20%的主控资源。

采用HMC5883L作为机身的电子罗盘。电子罗盘接到MPU6050的从IIC总线（auxda，auxdl）上，在初始化MPU6050时，设置成主IIC总线与从IIC总线直通，STM32可以直接通过主IIC总线访问从IIC总线，从而读取HMC5883L的数据。

数据更新模式采用硬件中断模式，即MPU6050和HMC5883L都有一个硬件中断引脚MP_INT和HM_INT，这样，能保证数据到来时间的准确，让CPU资源最大化利用。

由于电子罗盘是一个对电磁环境很敏感的元件，所以在布局时，尽量将HMC5883放在一个空旷的地带，周围不要有金属，附近不要有不要有大电流通过。

在这个问题的处理上，dji的Phantom系列飞行器已经有了很好参考，Phantom系列的电子罗盘是放在飞机的起落架上的，可以看出他们的工程师在设计飞行器的时候，考虑问题是很周全的。

电机驱动

无刷电机的操作相对来说是比较麻烦的，而有刷电机就是我们小时候玩的四驱车上的那种电机，接上电就能猛转，反着接它就反着猛转，就是这么简单。

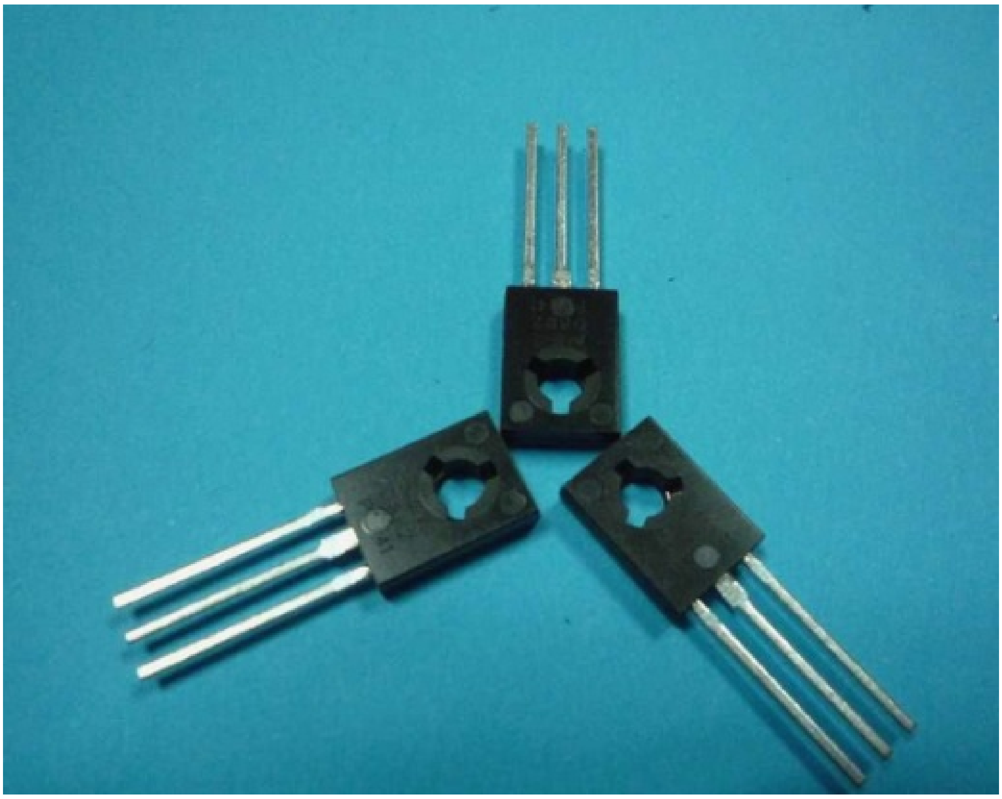
Crazepony使用的是有刷空心杯电机，所以电机的控制属于有刷直流电机控制。相对于无刷电调来说要简单很多，所以电调我们就默认指无刷电机的电调，而这里只用电机驱动来代替。Crazepony采用的是有刷空心杯高速电机，转速在3W转/分钟左右。要驱动有刷电机，很简单，只需要将信号的驱动能力增大，就能驱动有刷电机了。

那么选择什么元件来提供这样的特性呢?Crazepony的电机驱动IC选型经历了三级管，中功率管的失败，最后选用的是场效应管（即MOSFET）SI2302。

由于笔者完全是由于一种强烈的爱好选择了飞行器，最开始连有刷电机和无刷电机的物理结构区别都不知道，电调又是啥？傻傻分不清楚……

> 从一个几乎零基础的状态去选择电机驱动芯片，弯路是必须要走的，学费是必须要交的。曾以为书上学到的东西马上就能用，马上能转化为产品，后来发现真的是自己想多了。

最开始用的三极管作为电机驱动，采用很经典的共射电路“三极管工作在开关状态应该就行了吧？”画了用三极管驱动的PCB板，发现电机越转越慢，根本没劲。“也许是因为三极管扛不了大电流，好吧那我换个中功率管吧，集电极最大6A电流行了吧？”可以想象结果是不行的。

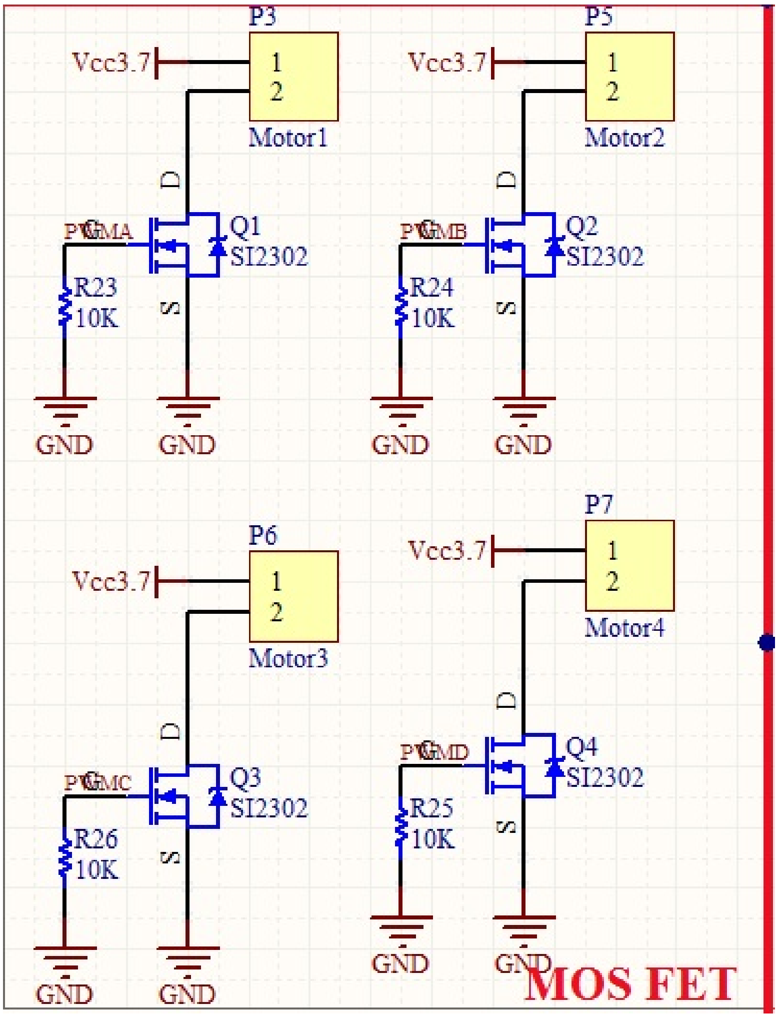


首先了解下为什么三极管作为简单的电机驱动是不可取的方案：

- 三极管作为一个古老的半导体先驱，它是以一个放大器件的姿态而出现的，它在线性区域特性集中，饱和与截止都是两种极端的工作状态，而作为电机驱动的话，我们只能选择它的这两种极端工作模式。

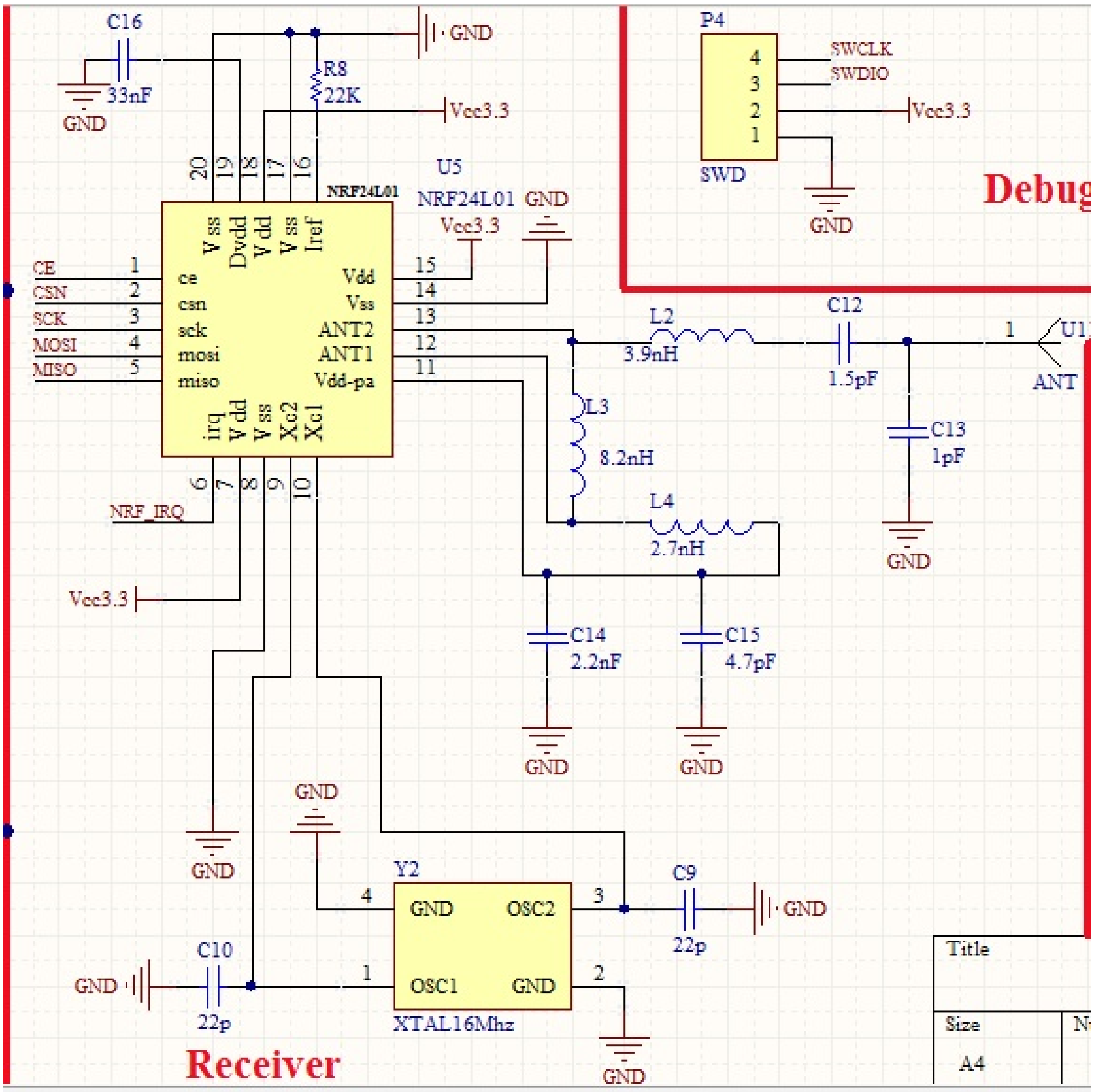
- 用三极管作为大电流负载的驱动管时，不得不考虑的是他自身的管压降对负载的影响，这是很严重的。自身耗散越来越大，电机和管子是串联关系，电池电压只有3.7V，电机就只能越转越慢了

在晶体管家族里面还有一种跟三极管特性互补的，所有特性都集中在开关状态的晶体管，场效应管，即MOSFET。通常的场效应管完全导通时，源漏极电阻都是mΩ级别的，即它自身的耗散非常小。用它做为驱动管再合适不过了。最终选择了一个SOT23封装的,导通电压Vgs<4v的场管（SI2302）,结果表现出了很好的驱动性能。



每个场效应管接一个大电阻下拉，目的是为了防止在单片机没接手电机的控制权时，电机由于PWM信号不稳定开始猛转。接一个下拉电阻，保证了场管输入信号要么是高，要么是低，没有不确定的第三种状态。那么电机也只有两种状态，要么转，要么不转。主控输出的是PWM波形，用于控制场效应管的关闭和导通，从而控制电机的转动速度。这就是crazepony电机驱动的原理。就是这么简单。

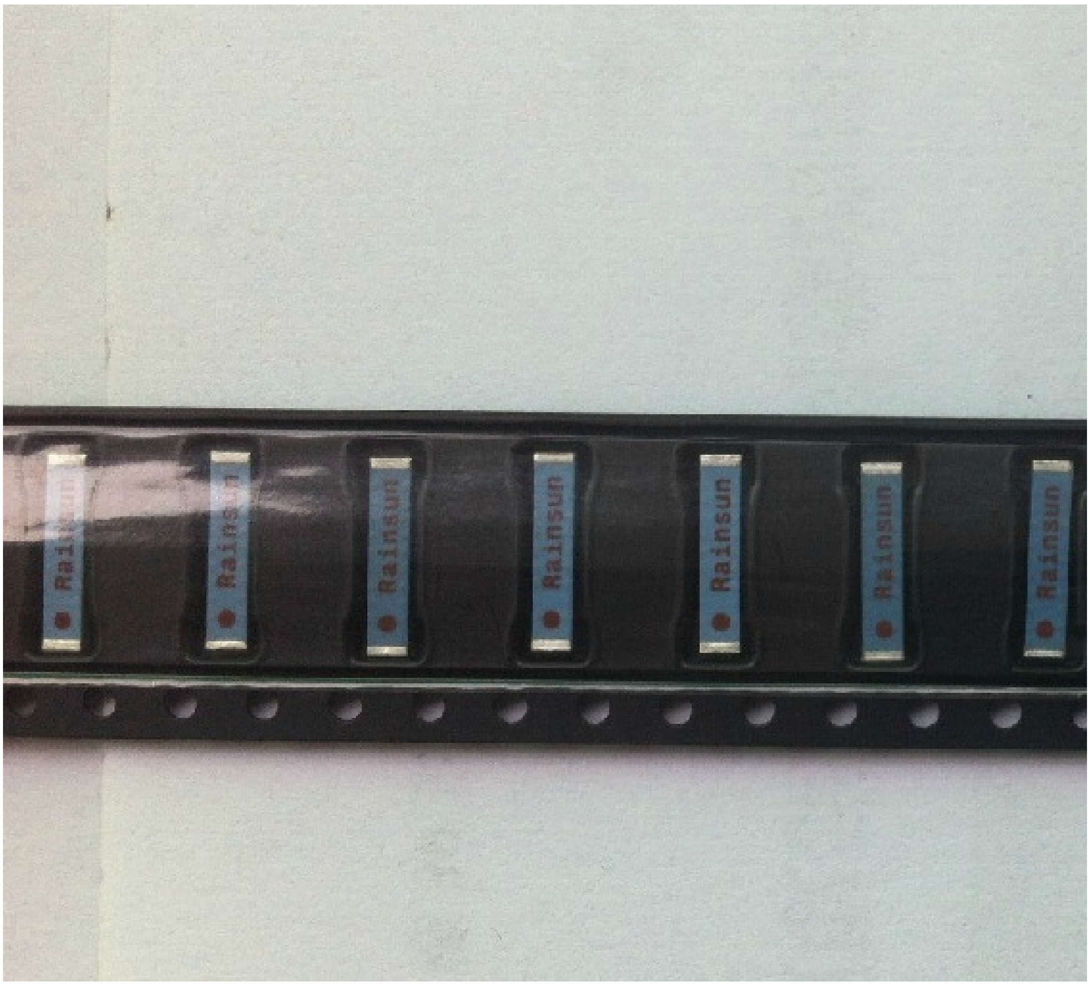
2.4G 通信



Crazepony在选择通信芯片上，也是选择了大家熟悉的芯片，NRF24L01。我相信很多朋友跟我一样，在大学阶段玩过最多的无线芯片就是它了，因为市面上有很多针对这个芯片延伸出来的各种通信模块，技术很成熟，也很稳定，性价比很高。

我查阅了NRF24L01的芯片手册，官方给出了整套硬件设计方案，包括天线的设计，PCB阻抗计算，以及底层的SPI驱动协议等等，资料很详细。

于是我们将这颗芯片画在了机身上面，在天线方面的选择上，我采用体积很小的2.4G陶瓷天线，这种天线比PCB天线占地面积小，对这个寸土寸金的小飞机而言，简直太棒了。



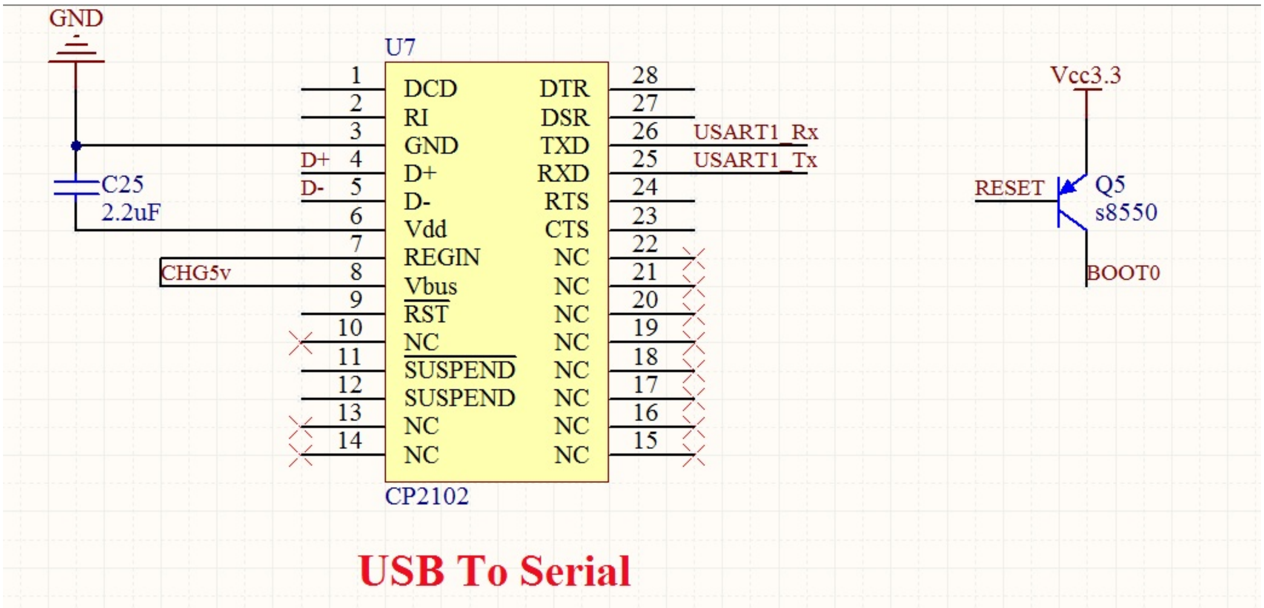
USB-Serial协议转换

前面已经说了启动模式的切换会在下载程序时用到。

Crazepony在设计之初，就考虑到了，一般用户PC机上是不会装像KEIL这种软件的，但是所有的PC机都有USB接口，那么在机身上集成一个USB-Serial协议转换芯片，那么就能够直接用一根micro USB的数据线（就是现在统一的安卓手机数据/充电线）对飞机进行固件（*.HEX）升级了，这种方式真是太棒了。

试想，你的电脑里原来都是跟你工作相关的文件或者是软件，今天你买来一台Crazepony，里面的固件虽然已经很优秀了，但是我们一直在完善它，突然有一天，我们告诉你，我们的固件升级了，你可以选择更新固件。那么，现在有两种方式供你选择来升级你的飞机：第一，你需要装一个叫做KEIL或者IAR或者其他类似的专业软件，还得在淘宝买一个你看都看不懂的jlink调试器。第二，你只需要将你的安卓手机线拿出来，插上电脑，打开我们的一键升级软件。这两种方式，我想大家都会选择第二种吧。

那么，要实现第二种方式，我们就需要用到启动方式的切换来实现。

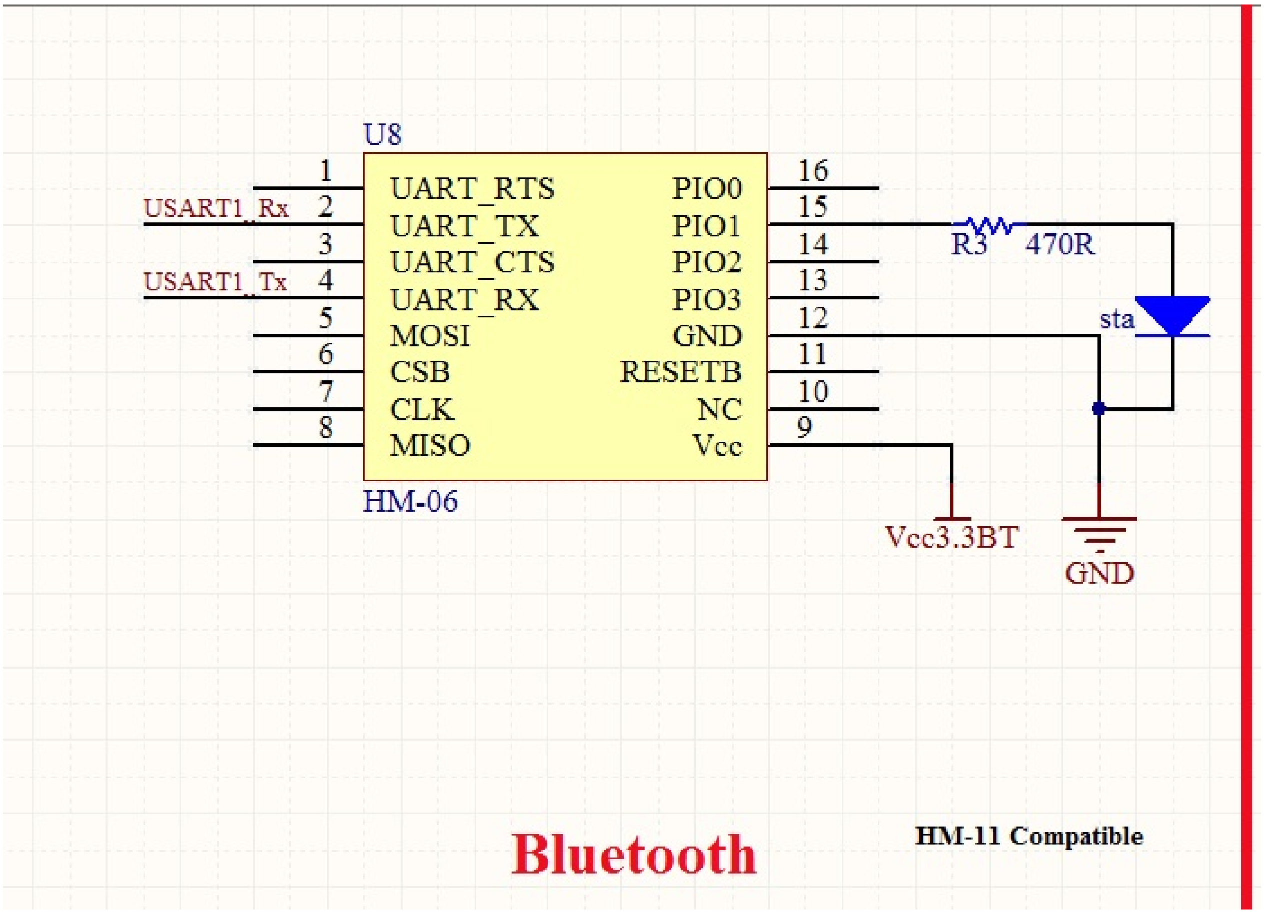


我们可以看到PNP型三极管Q5的基极是和复位按键连接的，PNP型三极管是基极低电平导通。Q5的发射极是和电源连接，集电极是和BOOT0连接。显然，BOOT0的电位高低，是跟复位按键是否按下直接相关的，同时我们有上面的铺垫也知道，BOOT0=1，BOOT1=1时，STM32从内部闪存开始启动，用于程序调试用。复位按键按下之前，BOOT0=0（内部下拉），复位按键按下以后，BOOT0=1，复位后，间隔四个时钟周期后采样BOOT0的电平，如果是高电平，那么进入调试模式，开始接收串口发送的程序代码，存放在内部的Flash中，下载完毕以后，软件复位，此时复位按键肯定已经弹起了，四个时钟周期以后采样BOOT0，此时是低电平，进入正常模式，这样就完成了一次ISP下载。

这样一来，机身和外部的有线接口就只有一根安卓手机的标配数据线micro USB线。它既是充电线，也是调参、烧写固件的数据线。这对大妈来说，想必操作也是很简单的吧。

有了USB-Serial的转换，那么我们的飞机和PC上位机的通信就有了必要的硬件基础。

蓝牙透传模块



为了让Crazepony能跟上智能机泛滥的步伐，同时也是为了增加它的适应性和降低成本的考虑，我们在原来的基础上增加了蓝牙透传模块，硬件焊盘上兼容蓝牙2.1和蓝牙4.0 BLE技术。就是说，同样一个Crazepony的裸PCB板，可以选择焊接蓝牙2.1和蓝牙BLE的模块。

蓝牙协议是很复杂的，要想去接触并试图一步一步的写出来，在我现有的时间下，几乎不太可能了。而将蓝牙透明传输成串口，这就很好的将Crazepony与安卓设备对接了，可以直接在安卓设备上面开发遥控app或者上位机。这想想也有点小激动呢，于是，我们真的这么干了，也成功了。

Crazepony的主要硬件介绍到这里就告一段落了，笔者刚结束学生时代，工程经验十分有限，硬件设计过程中肯定许多有不足和漏洞，希望广大读者指正，我们一定会积极接受意见并作出修订，希望把Crazepony做得更加完善，更加稳定，给这片土地上的广大爱好者提供丰富的资料和开发经验。

软件框架讲解

> 作者：nieyong

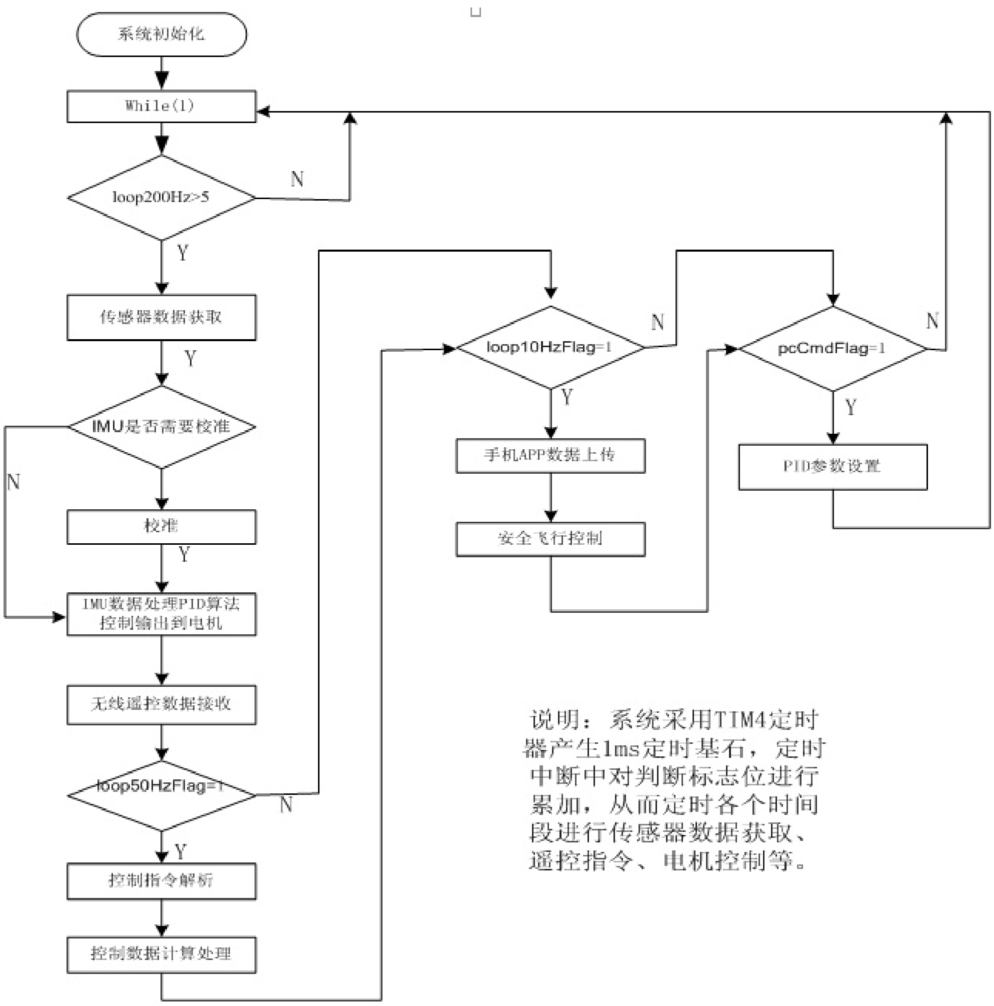
飞控源代码部分，都是属于一砖一瓦敲出来的。没有使用实时操作系统（RTOS），我们称之为裸机代码，托管在Github上，名字为crazepony-firmware-none，尾缀none表示未使用操作系统裸跑的意思。

那么，现在就结合裸机代码，来说说Crazepony的软件框架。

> 本文档以Crazepony 5.2版本为基础。Crazepony 5.0版本及以前的代码主要由马骏（CamelGo）完成。贡献者黄永祥在5.1版本中对飞控代码进行了重构，将Crazepony的稳定性推向了一个新的高度。贡献者Nieyong在5.2版本中对代码进行了整理。

软件流程图

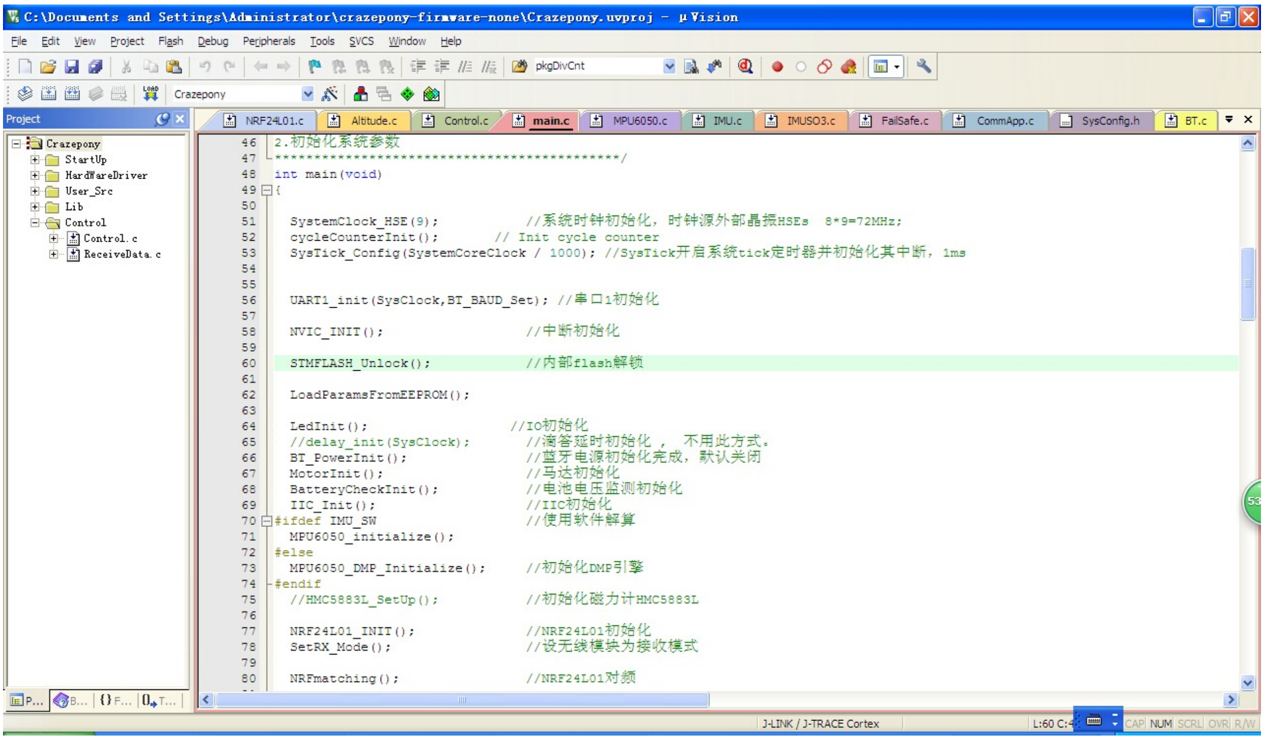
Crazepony软件流程如下图所示：



Craepony主控使用的是STM32芯片，没有上实时操作系统，依靠中断嵌套来完成整体功能。程序核心就是通过定时器，在主循环中通过不断查询判断各个条件，这样就产生了几个大小不一样的时间段，我们根据需要就可以完成以多大的频率扫描一次遥控器指令、多久更新一次传感器数据、多久更新一次控制等等飞控需要实现的功能，尽可能的利用主控的资源。

初始化

进入主函数之后就是STM32处理器及各个部分的初始化。

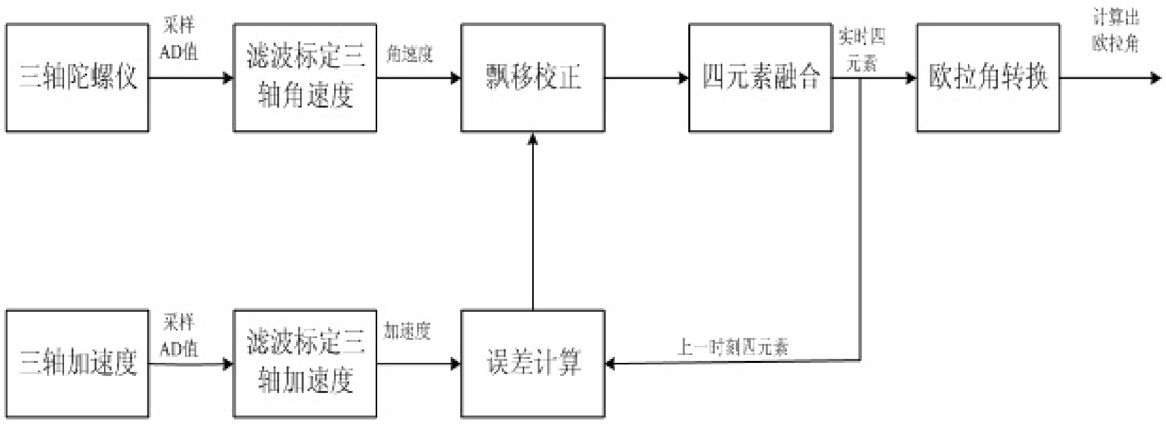


接下来就是进入主循环 `while(1)` 之中了，主循环也就是整个程序功能实现的关键，程序进入这里面就循环在里面运行了，当然中断会打断去运行中断服务程序运行完之后再回到这里运行。

主循环-100Hz循环

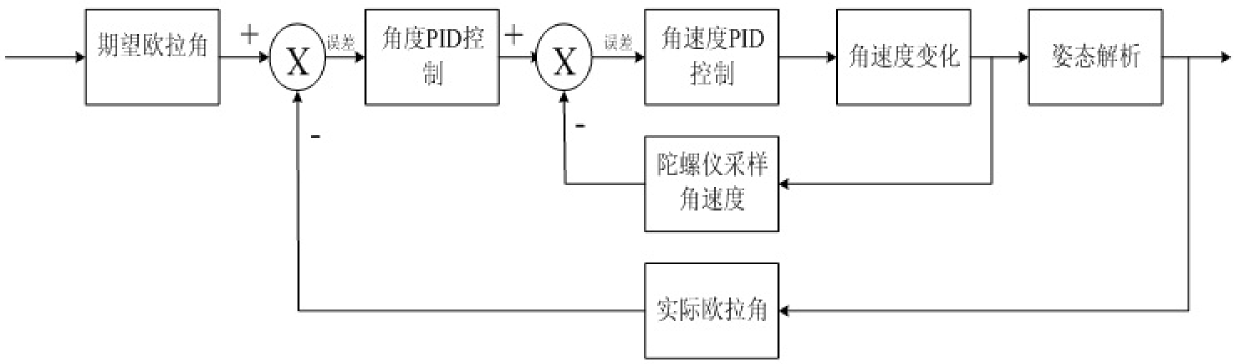
主循环体中首先有 `if(loop100HzCnt >= 10){}` 这个结构，其中loop100HzCnt这个变量是在TIM4中断服务程序中累加的，1ms累加一次，也就是说定时每10ms就去完成一次其中的工作。

那么100Hz需要做一次的工作是什么呢？读取mpu6050数据，气压计数据并进行整合。因为采用软解姿态，读取的数据为加速度计和陀螺仪的AD值，将数据进行标定、滤波、校正后通过四元素融合得到三轴欧拉角度。如下图。



加速度传感器采集数据容易失真，造成姿态解算出来的欧拉角错误，只用角度单环情况下，使系统很难稳定运行，因此可以加入角速度作为内环，角速度由陀螺仪采集数据输出，采集值一般不存在受外界影响情况，抗干扰能力强，并且角速度变化灵敏，当受外界干扰时回复迅速增强了系统的鲁棒性。

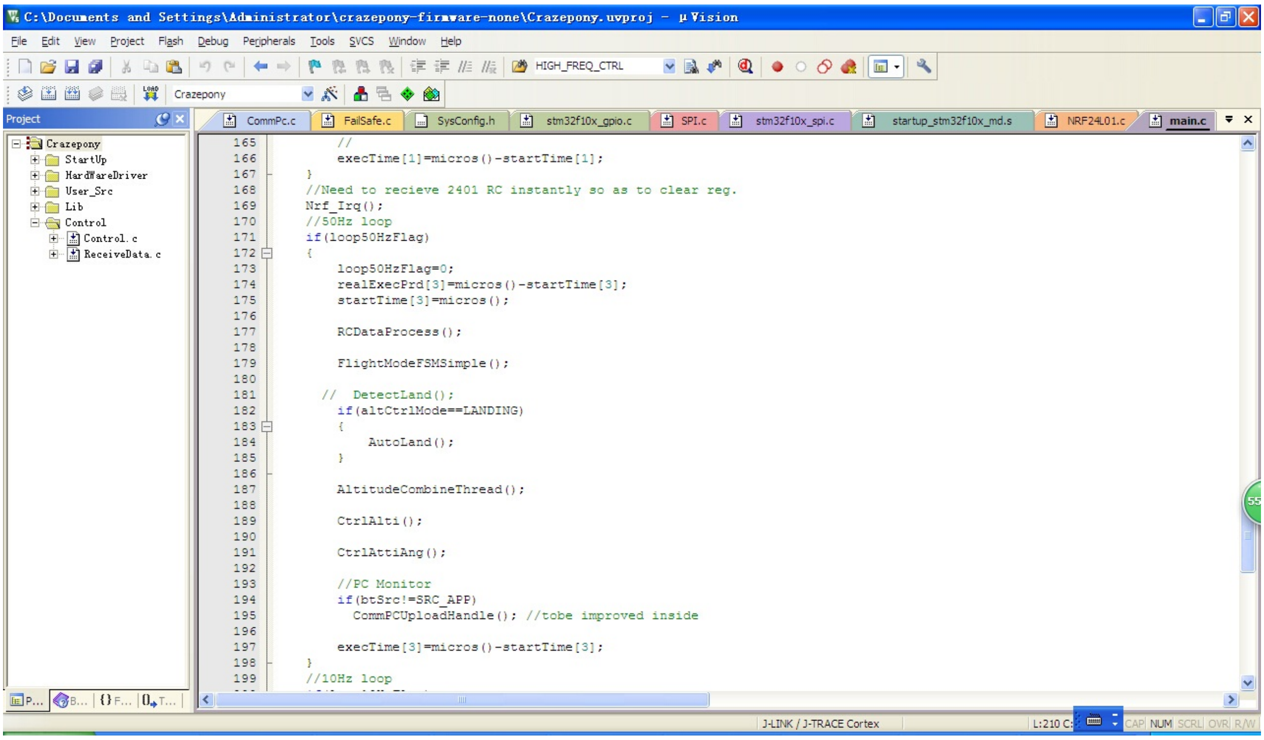
Crazepony采用双闭环PID控制，如下图所示。



角度作为外环，角速度作为内环，进行姿态双环PID控制。角度环的输出值作为角速度环的输入建立自稳系统。

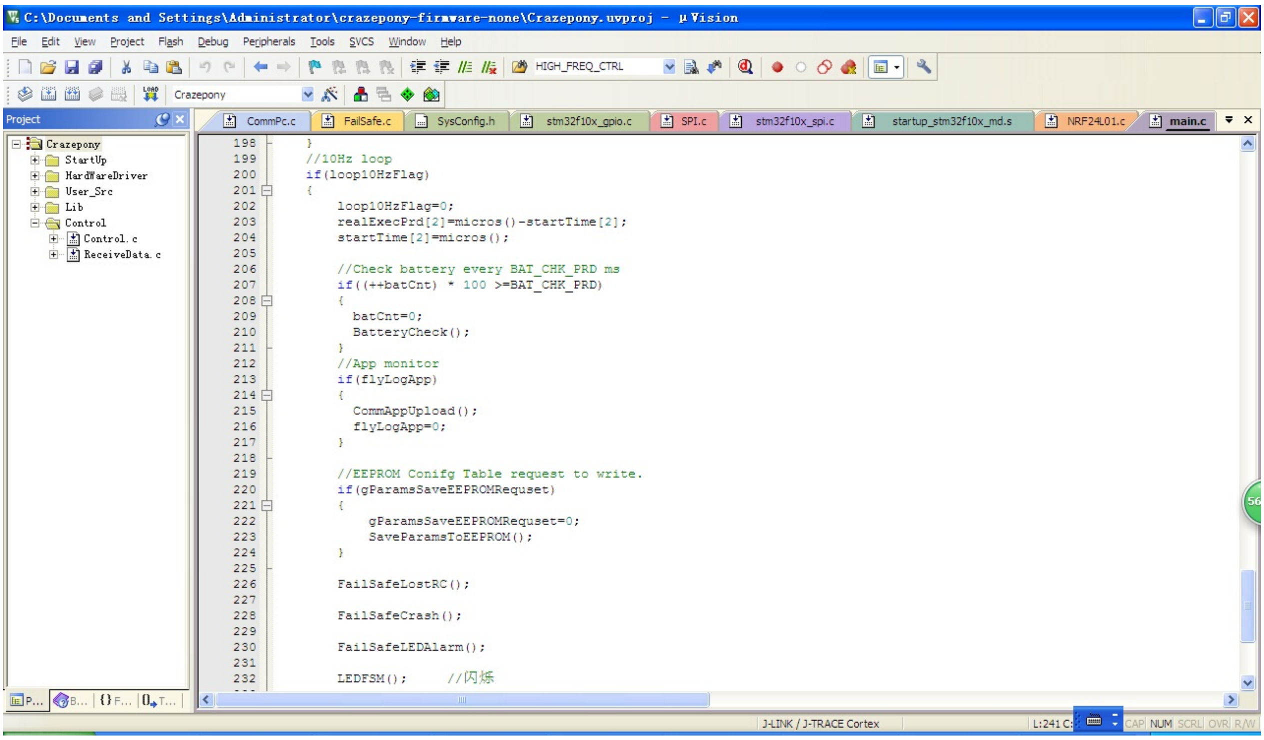
主循环-50Hz循环

在 `if(loop50HzFlag){}` 进入50Hz（20ms执行一次）循环。 `loop50HzFlag` 标志位是在TIM4中断中每20ms置位一次的，这里解析了收到的遥控器无线发送过来的指令，结合当前的姿态计算更新这些控数据给核心控制算法输出控制飞控，我们就可以控制飞控前进后退，上升下降等等操作了。如下图。



主循环-10Hz循环

同样的思路 `if(loop10HzFlag){}` 也就是以10Hz的频率去执行下面功能。在这里可以通过蓝牙向我们的手机APP传送一些飞控的姿态信息，然后查询飞控的电量没有足够的话就让飞控降落下来，查询高度啊超出可控范围也把飞控降下来，查询是否和遥控器失联啊，失联就降下飞控等等安全飞行的控制。如下图



最后就是if(pcCmdFlag)这个了，这是一个与上位机调试有关的东西，主循环查询这个标志位，标志位是由上位机发送过来的指令置位的，它主要是处理pc机发送过来的指令，PID参数读取，修改等等。

Crazepony软件开发经验总结

> 作者：CamelGo

关于Z轴负反馈的调试

虽然我们已经努力将机身参数设计成没有个体差异,但是还是避免不了一些电阻电容上有误差。直接体现在MPU6050输出的姿态AD值上有个体差异，下面谈一谈用户拿到飞机以后，怎样保证飞机起飞降落过程具有粘滞性的问题

什么叫粘滞性？就是说，飞机上升的时候，不会因为油门突然加大而突然猛升，也不会因为油门不小心拉低过快导致飞机快速跌落到地面，那么怎样才能达到这样的效果呢？

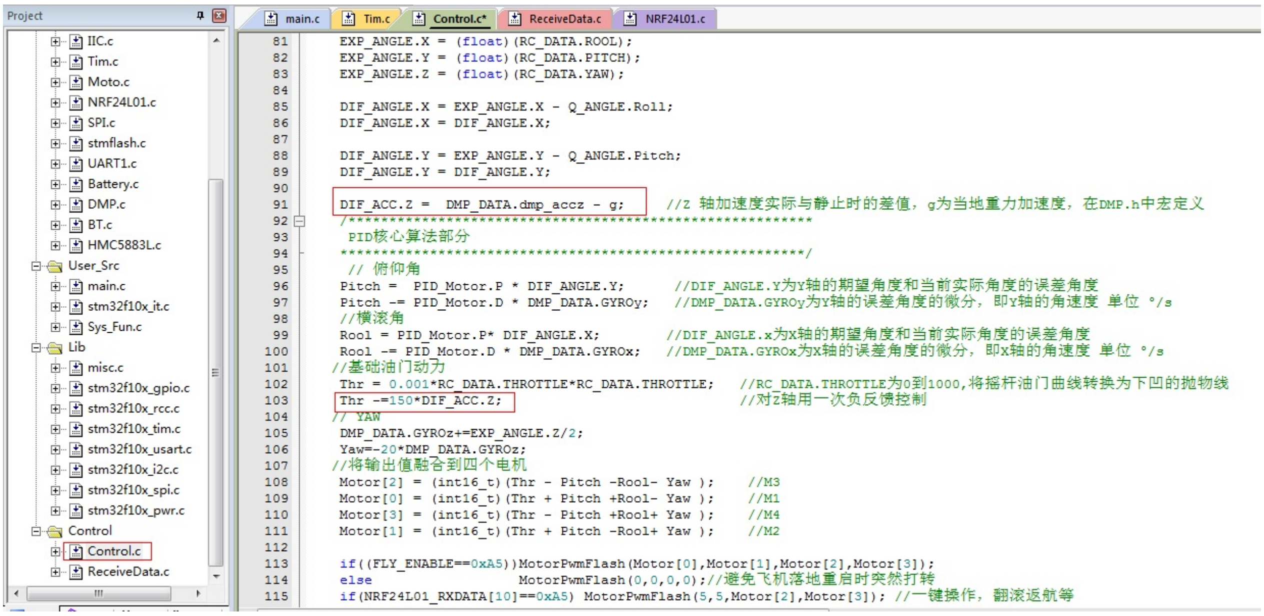
我们都知道，飞机在空中的任何一个动作或者姿态，都会被姿态传感器采集到，那么飞机上升和下降这个动作，对应的是哪个物理量在变化呢？Z轴的速度？还是飞机的高度？ 这两者都在变化，但是姿态传感器都不能直接给出，然而，Z轴的加速度是姿态传感器可以直接给出的

下面我们来理一下这个过程Z轴加速度是怎么变化的。

中学毕业的人都知道这样一件事情就是关于失重和超重，飞机加速上升这个过程，Z轴的加速度会大于静止时的加速度9.8m/s2，飞机加速下降这个过程，Z轴加速会小于9.8m/s2，这个物理量，可以由传感器直接得到，而这个物理量的变化，恰恰就能得到飞机是正在上升还是正在下降，那么上升的时候，程序控制它油门增加量不要变化那么快，下降的时候，程序控制它油门减少量不要减那么快。So…

学过大学相关控制理论的人都知道，这种控制方式叫做负反馈。

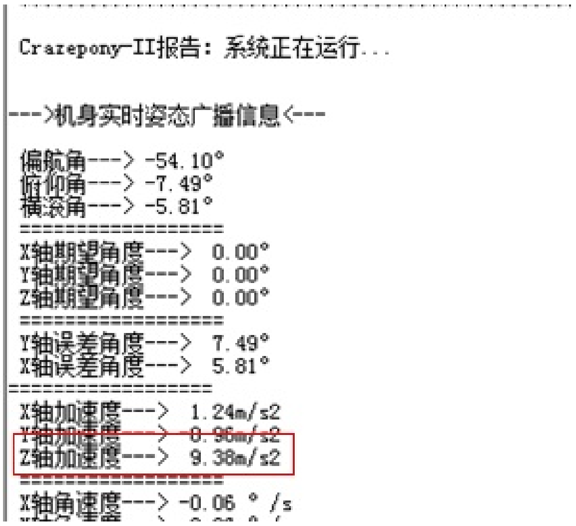
对应到程序就是这样的：



Thr是一个油门的全局变量，在Thr被遥控数据赋值以后，再对Thr进行一次减操作，减去一个误差量DIF_ACC.Z，DIF_ACC.Z是Z轴的当前加速度减去静止时的加速的差值。当飞机上升时，DIF_ACC.Z是一个正数，Thr减去一个正数，Thr就会比遥控实际赋值要小，体现出来的效果就是上升不会那么冲；同理，下降时DIF_ACC.Z为一个负数，Thr减去一个负数，相当于加上一个正数，Thr就比遥控实际赋值要大，体现出的效果就是下降不会很急。

于是，综上的描述，飞机的飞行体现出粘滞的效果，便于操控，也安全了许多

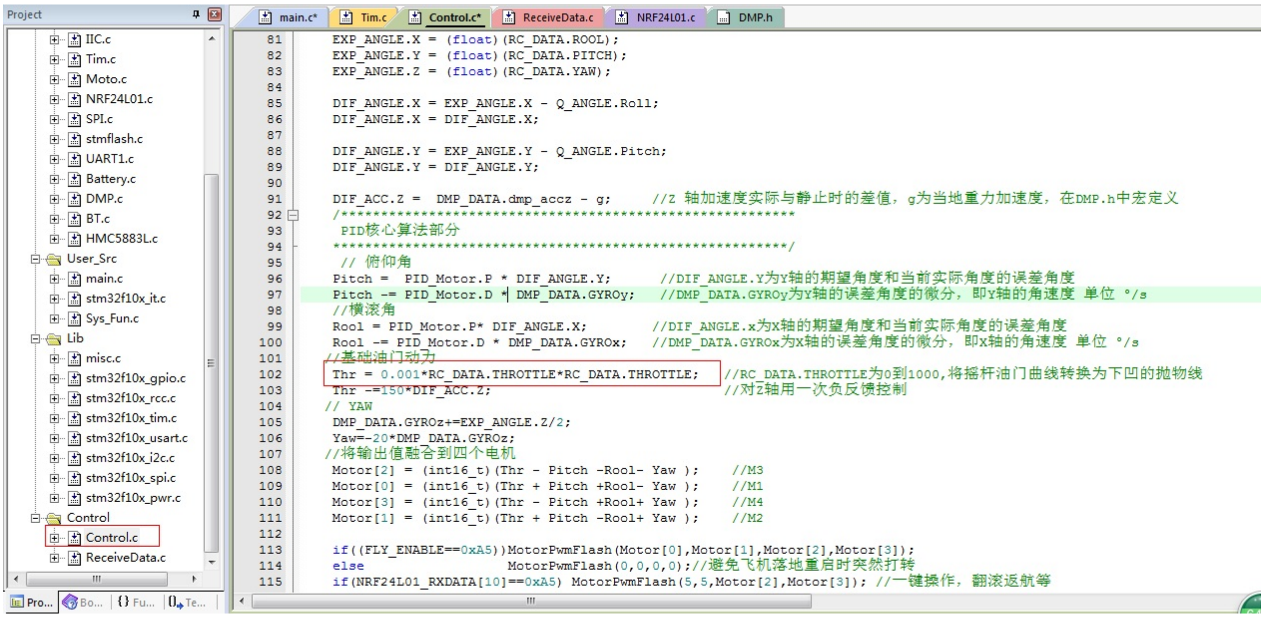
所以，在用户拿到手后，如果感觉操作起来不是那么方便，可以打开串口助手看看这里：



将Crazepony水平静止放置，然后看看这里打印的Z轴加速度值是多少，将源代码的静止重力加速度值g，修改成串口助手里看到的那个值就好了，这是属于一个初始人工校准过程，我们后续会考虑实现一个学习算法来替代这个过程，它将不会再需要用户介入，你只需要开一个电源开关，学习算法会自我学习，将学习得到的最优参数存放在内部的EEPROM中。

关于油门曲线

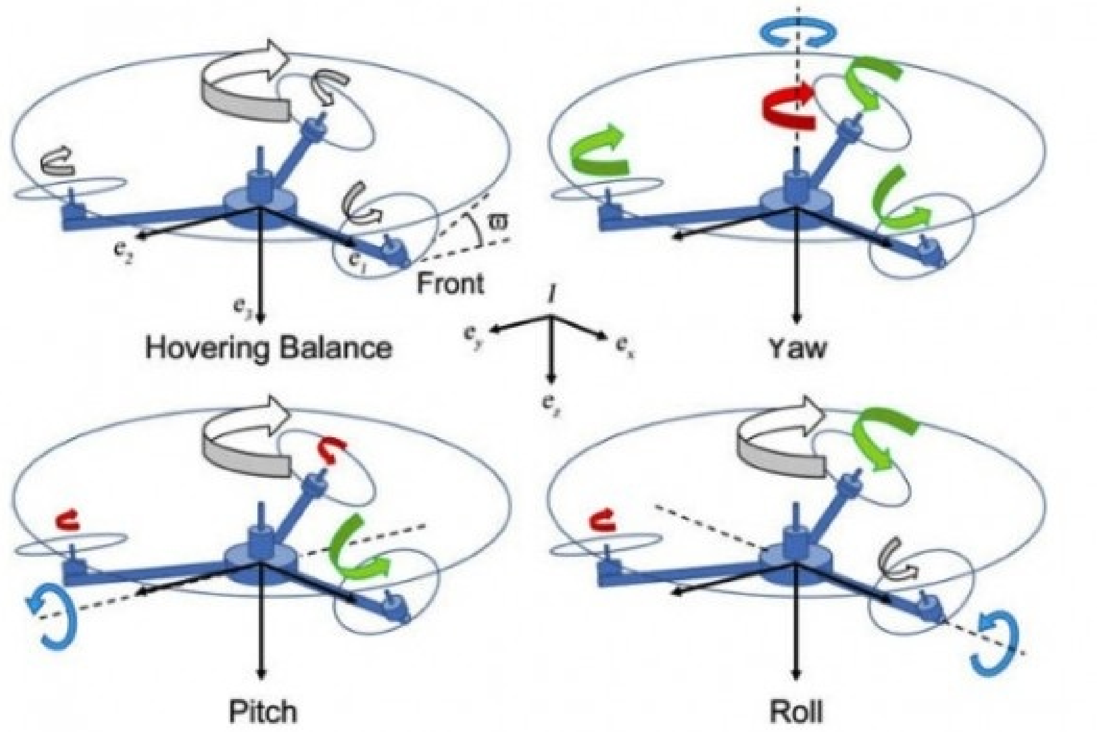
在Crazepony最初的几个版本中，我并没有油门曲线这个概念，遥控给多少，就直接给PWM输出，根本不做任何可能的处理。在后来的飞行体验中，慢慢发现。油门摇杆的前半程，很容易突然推很多，导致飞机一下冲很高。于是，在跟大家的交流中发现，将遥控送给的油门值曲线化以后再给PWM输出，这样的话。就能避免这个问题了。



很显然，我将油门曲线做成了一个下凹的抛物线。抛物线方程为： y=0.001*X*2，即*Thr=0.001*（RC_DATA.THROTTLE）2，RC_DATA.THROTTLE的取值范围是0-999，Thr的取值范围也是0-999，这样一来，将遥控的油门摇杆取值范围，跟飞机实际得到的油门量做了一个一一映射。最大值和最小值都是一样的，但是处理后的油门曲线是一条下凹的曲线，和原来线性对应相比，优势大家自己实际体会吧

开发指南

本章介绍如何在windows/linux下搭建开发环境，如何编译源代码，如何下载到主控上进行调试等二次开发的步骤。对于想对Crazepony进行二次开发的小伙伴，能够提供非常完备的指导。



Windows下开发环境搭建-裸机版本

> 作者：nieyong

IDE开发工具Keil v5.10下载地址：Crazepony百度云网盘Keil-uVision5-MDK5-12.zip（5.2及以后版本）。

IDE开发工具Keil v4.10下载地址：Crazepony百度云网盘Keil-uVision4-MDK4-12.zip。

飞控源代码下载地址：Github项目crazepony-firmware-none。

CP2102驱动及ISP烧录工具下载地址：Crazepony百度云网盘开发工具

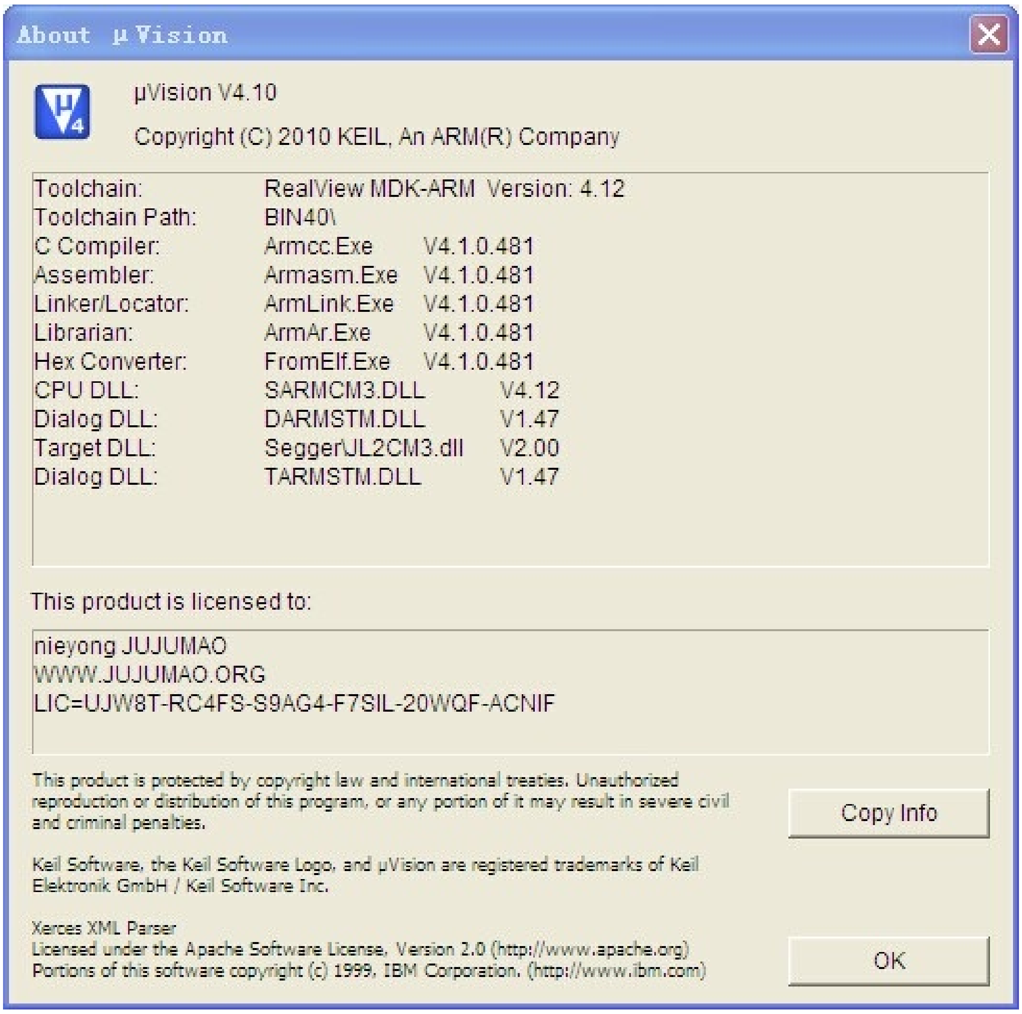
环境配置说明

裸机版本是指没有使用RTOS的主控固件版本，由于没有使用RTOS，程序代码显得更加简单明了，开发环境配置简单，非常适合初学者使用。

裸机版本的代码使用Keil进行开发编译，下面介绍编译环境的搭建和编译过程。在crazepony 5.1版本及以前，使用keil 4.10进行开发。但是由于其不支持utf-8编码，所以从5.1版本之后都会使用keil 5.10版本进行开发。

Keil 4的安装和破解

由于主控使用的是ARM Cortex-M3内核的STM32，所以必须使用带有ARM编译工具链的Keil 4。网络上很多只有几十兆大小的的Keil 4可能只支持C51开发。Crazepony百度云网盘提供带有ARM编译工具链的Keil 4安装文件下载，带有破解注册机和破解说明文档。用户自行下载安装，可以看到该Keil版本为v4.10。



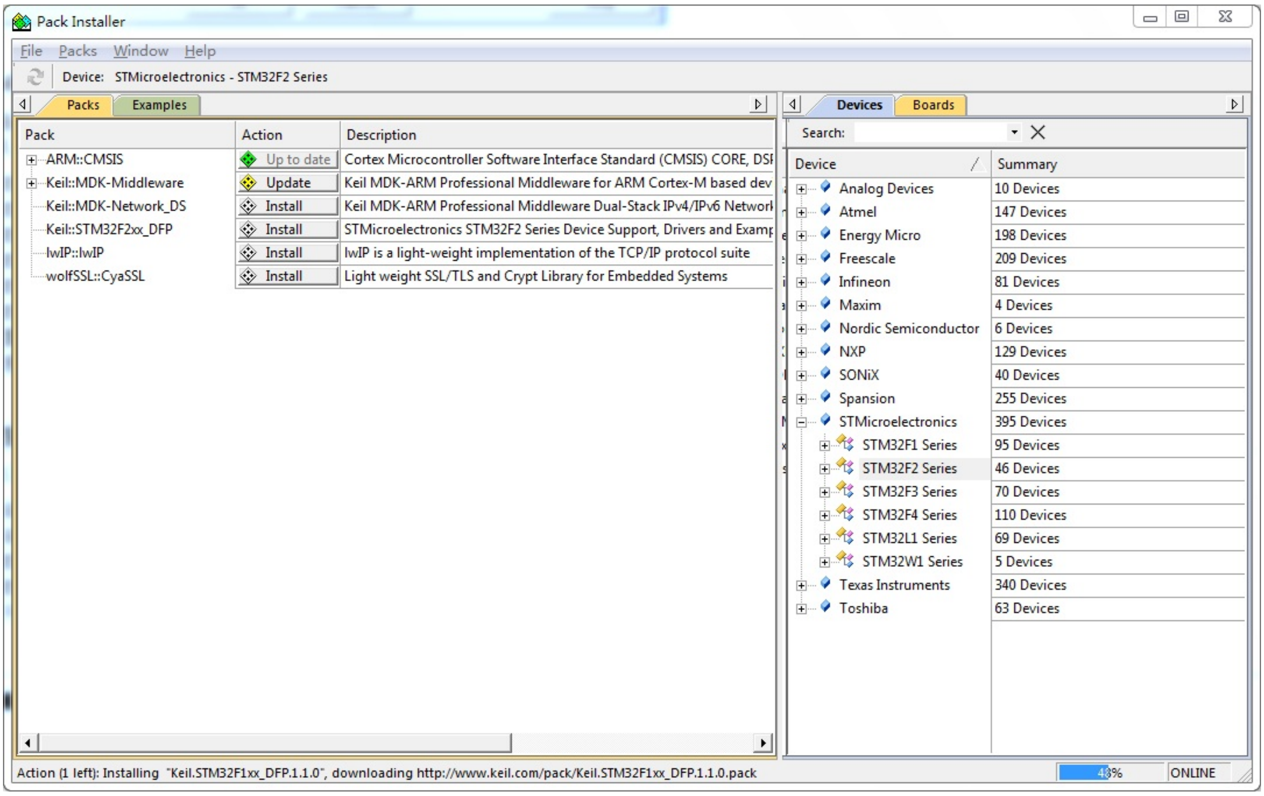
该安装文件在Windows XP，32位系统上测试通过。已知存在的问题：

- 不支持utf-8编码。utf-8编码的中文在Keil 4下面会显示为乱码，所以这部分固件代码没有使用utf-8编码。没有找到能够配置支持utf-8的地方，很可能是Keil版本比较老，或者破解导致无法支持utf-8编码。

Keil 5的安装和破解（推荐）

最新crazepony版本都使用keil 5.10进行开发调试，并且文件使用utf-8编码。crazepony百度云盘提供keil 5.10安装文件下载。并且可以注册破解。

Keil 5的安装文件是Keil 4的几倍，很多组件与模板都需要分开来安装，安装稍微比较麻烦。在安装完主程序之后，会弹出Pack Installer的对话框，用于被开发芯片对应模板的安装。crazepony使用的STM32F103T8主控芯片，所以从右边选择STM32F1 Series，在左侧点击安装。



源代码下载

主控固件裸机版本，裸机指没有使用实时操作系统RTOS。由于没有使用RTOS，所以代码更加简单明了，适合初学者使用。

Crazepony项目是开源的，所有代码都托管在Github的Crazepony项目下。裸机源代码放置在crazepony-firmware-none下，命名中的none表示不使用RTOS。

可以直接使用git获得源代码，也可以从crazepony-firmware-none release下载各个版本的裸机源代码。

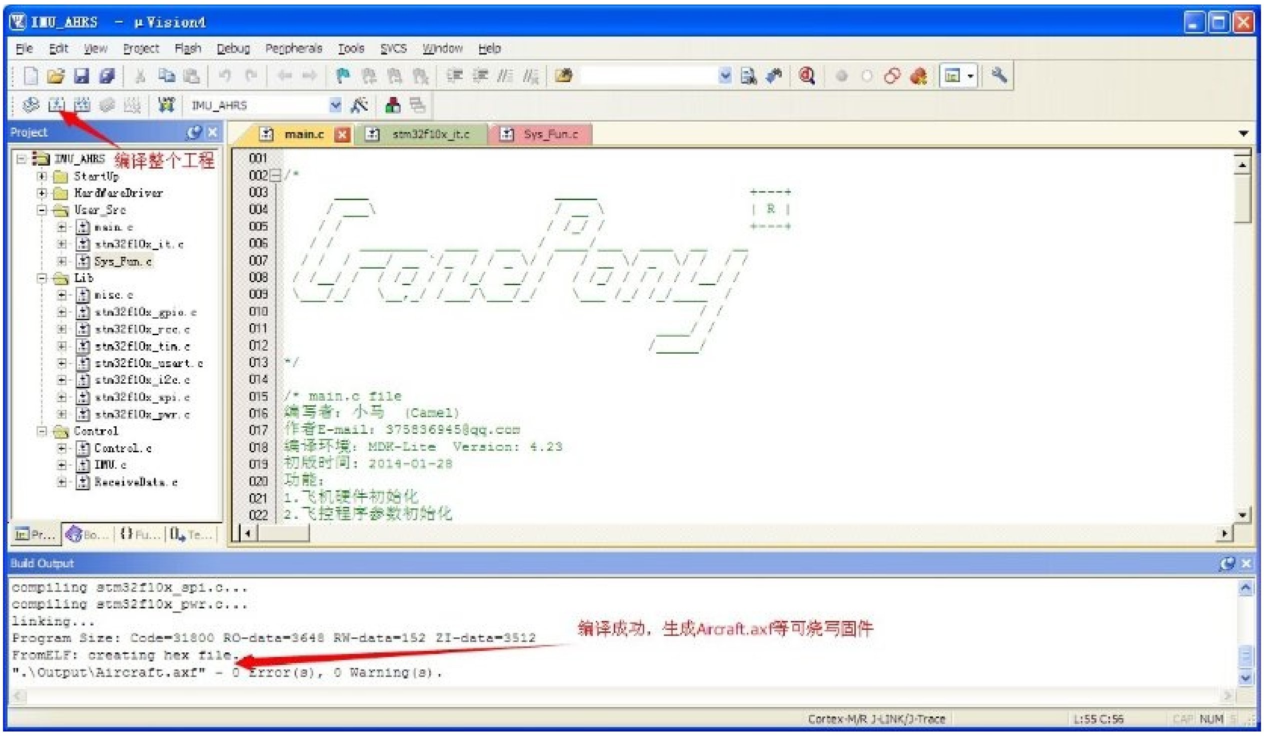


源代码导入和编译

在源代码中，有三个文件为Keil工程文件。

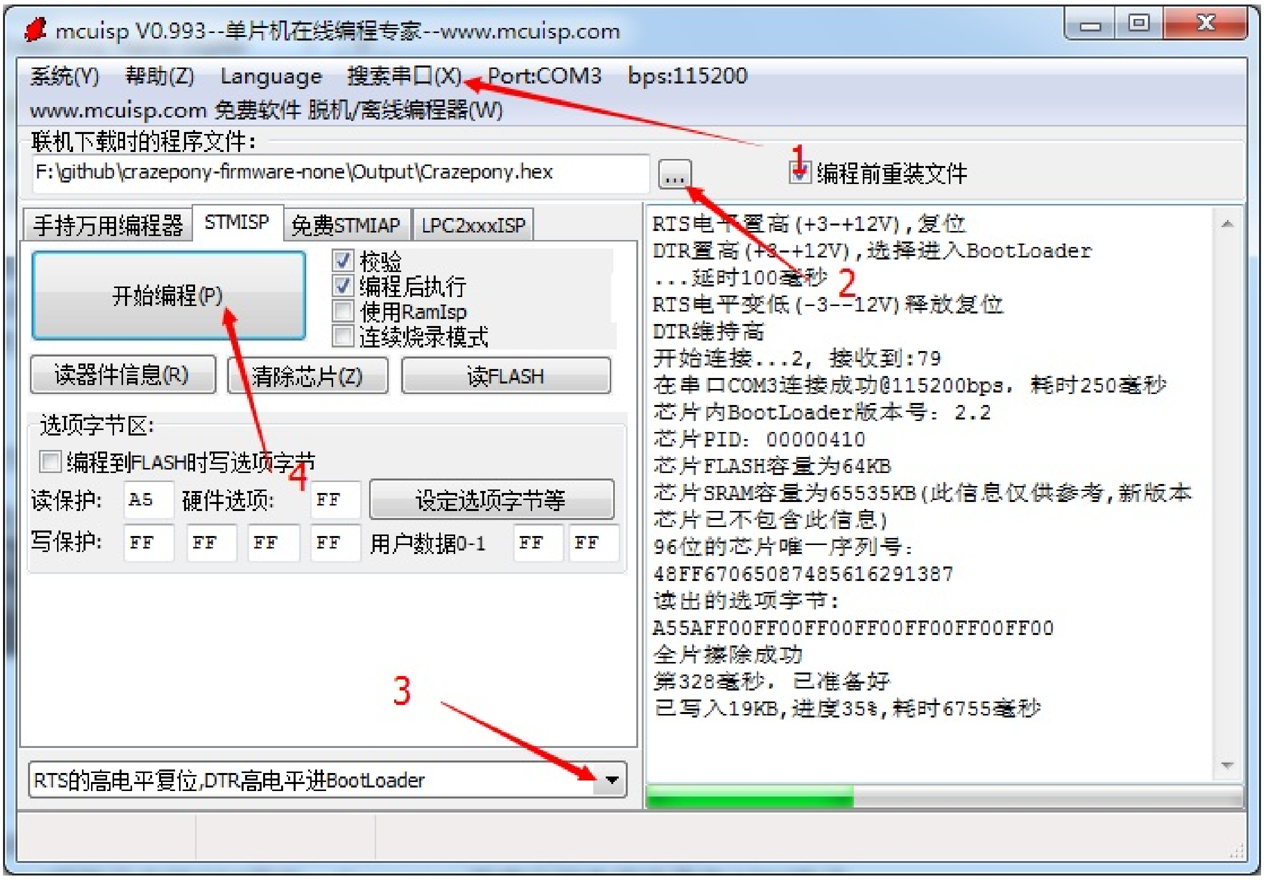
```
Crazepony.uvopt
Crazepony.uvproj
Crazepony.dep
```

以Keil 5为例，点击Project栏目下Open project，打开代码解压所在文件夹选中Crazepony.uvproj，此时已将代码项目工程所有文件导入Keil 5，如图所示：



点击左上角的编译按钮，编译整个项目，在工程下生成Output目录，目录中Crazepony.hex就是可以烧写到STM32的固件。

STM32固件的调试和烧入可以使用J-Link或者ST-Link等调试器进行开发，详见J-Link的使用及常见问题。Crazepony也支持USB口烧入，即采用ISP下载，操作简单。只需要安装cp2102驱动程序，使用一根Mini USB数据线连接电脑。详见固件烧写，烧录界面如图所示:


mcuisp V0.993--单片机在线编程专家--www.mcuisp.com
系统(Y) 帮助(Z) Language 搜索串口(X) Port:COM3 bps:115200
www.mcuisp.com 免费软件 脱机/离线编程器(W)
联机下载时的程序文件:
F:\github\crazepony-firmware-none\Output\Crazepony.hex
编程前重装文件
手持万用编程器 STMISP 免费STMIAP LPC2xxxISP
开始编程(P)
校验
编程后执行
使用RamIsp
连续烧录模式
读器件信息(R) 清除芯片(Z) 读FLASH
选项字节区:
编程到FLASH时写选项字节
读保护: A5 硬件选项: FF 设定选项字节等
写保护: FF FF FF FF 用户数据0-1 FF FF
RTS的高电平复位,DTR高电平进BootLoader
RTS电平置高(+3-+12V),复位
DTR置高(+3-+12V),选择进入BootLoader
...延时100毫秒
RTS电平变低(-3--12V)释放复位
DTR维持高
开始连接...2, 接收到:79
在串口COM3连接成功@115200bps, 耗时250毫秒
芯片内BootLoader版本号: 2.2
芯片PID: 00000410
芯片FLASH容量为64KB
芯片SRAM容量为65535KB(此信息仅供参考,新版本芯片已不包含此信息)
96位的芯片唯一序列号:
48FF67065087485616291387
读出的选项字节:
A55AFF00FF00FF00FF00FF00FF00FF00
全片擦除成功
第328毫秒, 已准备好
已写入19KB,进度35%,耗时6755毫秒
1
2
3
4

Windows下开发环境搭建

> 作者：韦晟敢

环境配置说明

目前采用Keil+gcc+jlink是最好的编译与调试方法。Keil内部有自己的编译工具链，但是Keil自带的编译器不支持目前代码的许多语法，所以建议使用gcc工具链。等代码稳定未来可以直接移植到Keil自带的编译器上。

安装arm-none-eabi gcc工具链

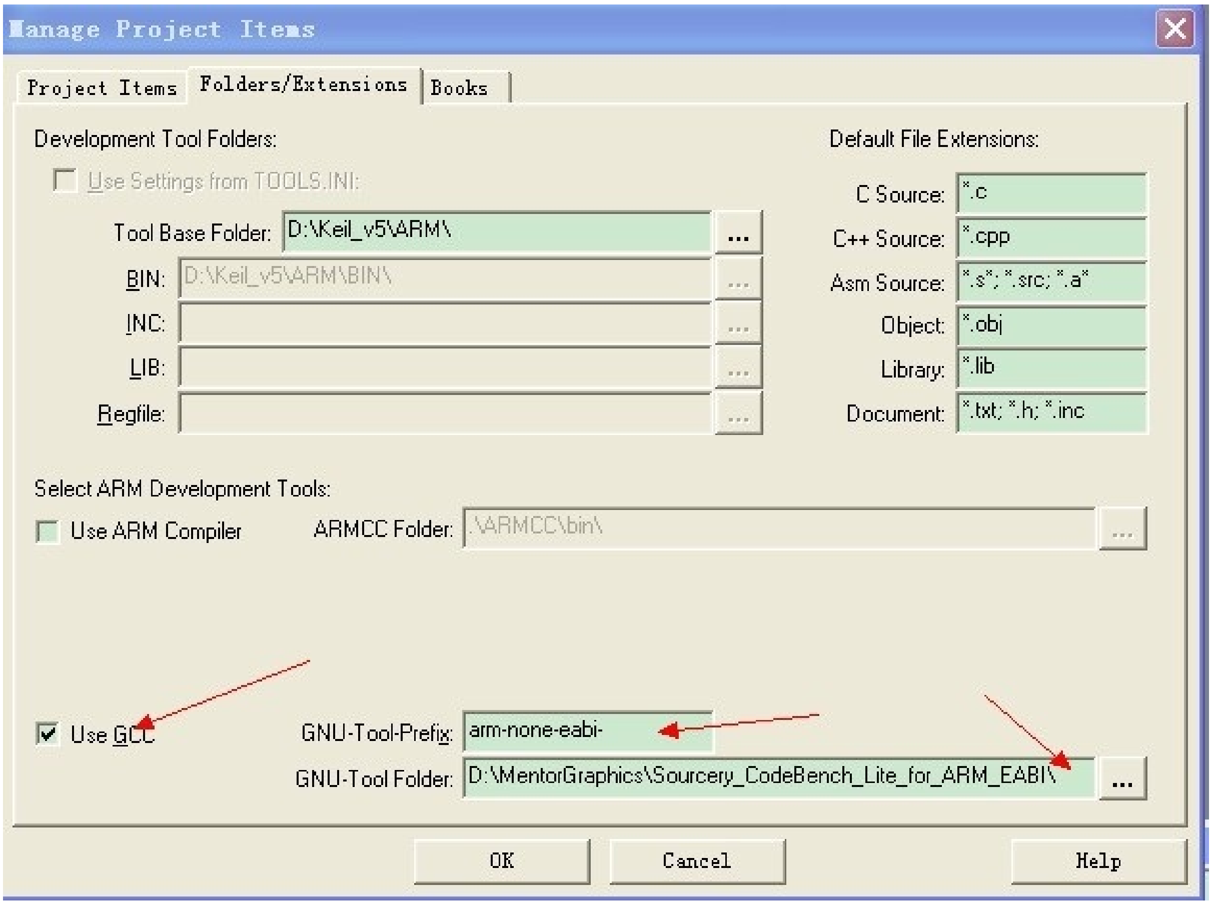
arm-2013.11-24-arm-none-eabi的下载地址

- 为了后面配置的方便，在其中的"Next"菜单中选择“添加参数到环境变量” (菜单提示的是英文)
- 路径请注意中间不要有空格。比如我的根路径是：D:\MentorGraphics\Sourcery_CodeBench_Lite_for_ARM_EABI

使用Keil4

若Keil4能配置成功，选择Keil4将是最方便的办法。安装Keil4并配置gcc的方法：

- 打开工程
- 点击菜单 Project->Manage->Component,Envrenment,Book。选择Folders/Entensions，选择勾中gcc
- 把路径设置为工具链的根目录，比如我的环境设置为：D:\MentorGraphics\Sourcery_CodeBench_Lite_for_ARM_EABI
- 如图所示：



使用Keil5

若Keil4通过以上配置，编译工程的时候提示找不到gcc或认不出来编译文件的名字，那么请安装Keil5。Keil5的安装文件是Keil4的几倍，很多组件与模板都需要分开来安装，安装稍微比较麻烦。

- 下载并安装：mdk510.exe
- 下载并安装：mdkcm510.exe
- 下载并安装：Keil.ARMCortex_D...0.0.1.pack Cortex相关模板文件
- 下载并安装：Keil.STM32F1xx_D...1.0.4.pack STM32F1xx相关模板
- 下载破解程序 打开keil5-->FILE-->Lisence Managerment 用破解程序生成序列号并输入到keil5里。破解成功。
- 下载：http://url.cn/O3oSQL 解压并替换keil5根目录开始的C:\Keil_v5\ARM\Segger压缩包里的两个文件，以进行jlink调试：
- 配置gcc如同Keil的方法。
- 注意jlink的驱动：win7 x64与win7/winxp x86可能不一样（TODO提供链接下载）

使用原生态的make编译

使用原生态的make编译也很方便，并且也可以通过keil来下载并调试

- 安装arm-none-eabi工具链 修改工具链根目录下的cs-make.exe为make.exe，cs-rm.exe为rm.exe
- 安装python2.7并把其加入到环境变量
- 下载MyGit，并得到Git的windows版工具
- 下载crazyflie的默认python-client代码，与此版本代码在同一级目录
- 打开MyGit，cd到项目代码目录并： `make`

STM32开发环境搭建-linux

> 作者：nieyong

Crazepony飞控主板使用STM32作为主控制器，下面主要介绍如何在ubuntu系统下进行STM32固件的开发/编译/下载。

对STM32开发需要在ubuntu系统下搭建arm嵌入式开发的交叉编译工具链。测试的系统为ubuntu 14.04。

交叉编译工具链安装方法一

STM32处理器使用gcc-arm-none-eabi-交叉编译链。使用wget下载交叉编译工具链到本地，并且解压缩到 `~/bin/` 目录下。

```
wget https://launchpad.net/gcc-arm-embedded/4.7/4.7-2013-q1-update/+download/gcc-arm-none
mkdir ~/bin
```

修改~/.bashrc文件，将交叉编译工具链的路径添加到当前用户的默认搜索路径下。并且使用source命令使其马上生效。

```
echo -e "\nPATH=\$PATH:$HOME/bin/gcc-arm-none-eabi-4_7-2013q1/bin" >> ~/.bashrc
source ~/.bashrc
```

可以使用type命令检测交叉编译工具是否安装成功。例如检测其 `arm-none-eabi-gcc` 命令，可以看到其在刚才安装的~/bin目录下检测到。

```
type arm-none-eabi-gcc
arm-none-eabi-gcc 是 /home/ny/bin/gcc-arm-none-eabi-4_7-2013q1/bin/arm-none-eabi-gcc
```

交叉编译工具链安装方法二

方法一需要手动下载gcc-arm-none-eabi-工具链压缩包，解压缩，然后手动配置工具链的地址系统默认路径下。该方法适合几乎所有的Linux系统，例如fedora/ubuntu/debian等。对于ubuntu用户，可以直接使用apt包管理工具进行安装。

- 由于gcc-arm-none-eabi-没有包含在ubuntu默认支持的包内，需要添加一个支持gcc-arm-none-eabi-的源：

```
sudo add-apt-repository ppa:terry.guo/gcc-arm-embedded
```

- 更新添加的源：

```
sudo apt-get update
```

- 安装gcc-arm-none-eabi-工具链

```
sudo apt-get install gcc-arm-none-eabi
```

使用type命令检查是否安装成功，如果出现下面的提示，那么就说明安装成功，可以使用。

```
$ type arm-none-eabi-gcc
arm-none-eabi-gcc 是 /usr/bin/arm-none-eabi-gcc
```

arm-none-eabi说明

arm相关的交叉编译工具链由很多，例如arm-linux-eabi，arm-none-eabi。STM32处理器使用的是arm-none-eabi。

```
arm-none-eabi和arm-linux-eabi等的区别：
arm-linux-eabi 用于编译arm linux内核代码。
arm-none-eabi 不指名操作系统，可以是linux，也可以是vxworks等，arm-none-eabi 不包含 __linux__ 等
arm-none-linux-eabi 用于Linux的，使用Glibc。
```

硬件开发工具

Crazyflie电路原理图使用的KiCad，KiCad以一个开源的EDA工具，并且跨平台，在windows和linux下都能够使用。

KiCad的home page：http://www.kicad-pcb.org/。在ubuntu下的安装：

```
sudo apt-get install kicad
```

Crazepony Android APP开发和配置

> 作者：nieyong

Crazepony的Android客户端源代码也是托管在github的crazepony项目下，点击这里查看。可以直接下载zip压缩包，也可以通过git clone到本地。

```
git clone git@github.com:Crazepony/crazepony-android-client-none.git
```

> Crazepony Android APP在5.0版本使用ADT Bundle进行开发。5.1版本以及以后版本使用Android Studio进行开发。

ADT Bundle开发环境搭建

Crazepony 5.0版本的Android APPP开发环境使用Google Android提供的一站式开发包ADT Bundle。ADT Bundle提供了Android APP开发所需的所有工具，包括Eclipse IDE，ADT (Android Developer Tools) 插件，java库等等。

> 下载地址：http://developer.android.com/sdk/index.html。该下载地址现在已经不再提供ADT Bundle下载。

下载解压缩到任意目录，就可以使用，无需安装。

Android Studio开发环境搭建

Google已经提供Android集成开发环境Android Studio，不再推荐使用原来集成Eclipse IDE，ADT，Java库等ADT Bundle。所以Crazepony 5.1版本以及以后版本使用Android Studio进行开发。

> Android Studio官网下载地址：http://developer.android.com/sdk/index.html。

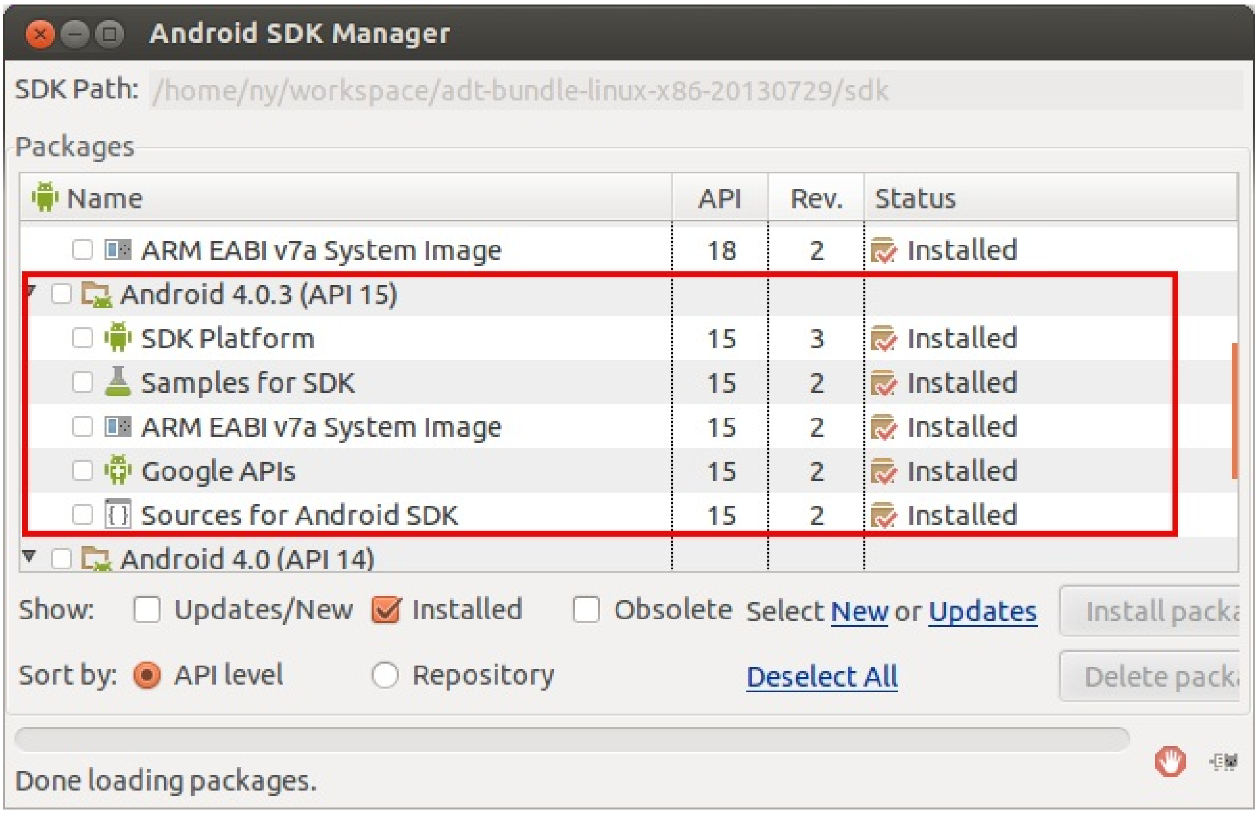
在中国国内提供百度云盘下载。

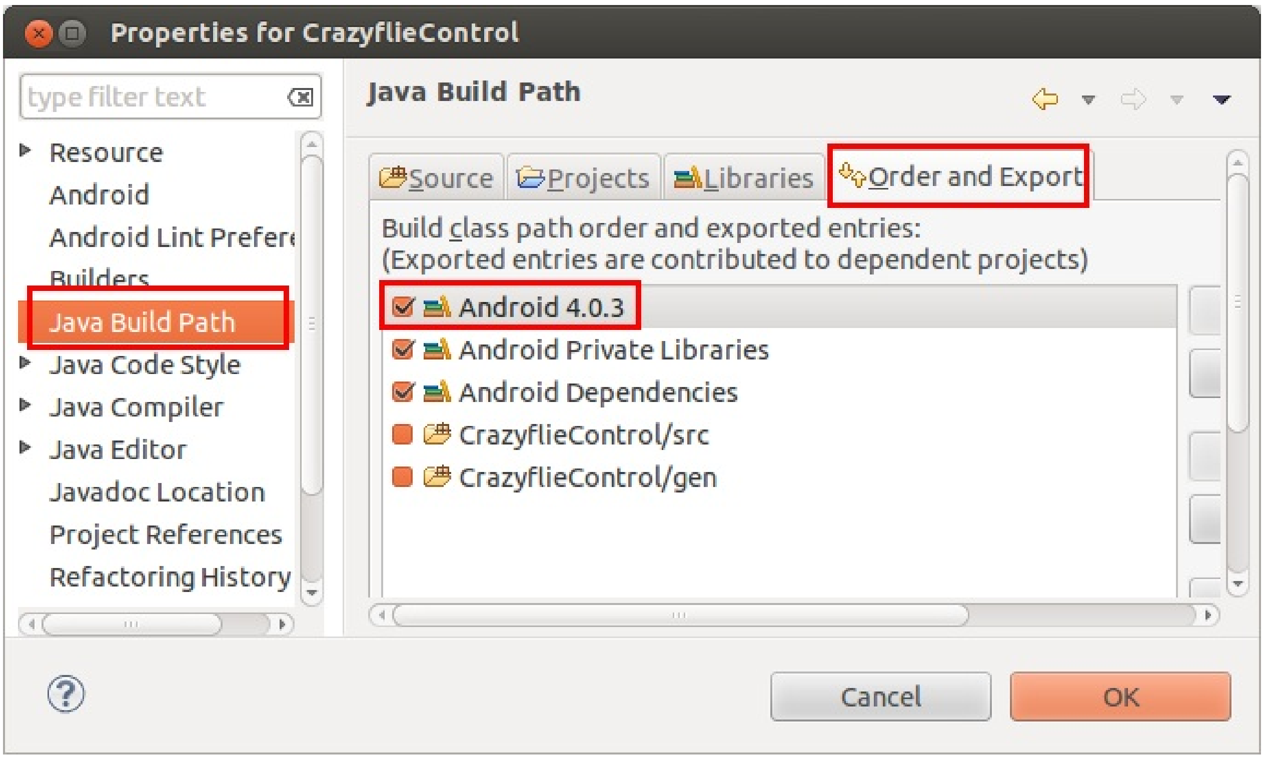
> 百度云盘下载地址：点击下载， 提供了Android Studio及对应的JDK下载。

首先安装下载的jdk-7u67-windows-i586.exe，然后再安装Android Studio开发环境。

配置说明

Crazepony的Android客户端默认编译的Android SDK API版本为15（对应Android 4.0.3），支持的Android版本为4.0.0或以上。





USB固件烧写

> 作者：nieyong

编译好STM32的代码，得到hex格式的固件文件，如何烧写到主控中？

Crazepony将SWD调试接口预留出来，可以使用J-Link或者ST-Link等调试器进行开发，详见J-Link的使用及常见问题。Crazepony也支持USB口烧入，即采用ISP下载，操作简单。只需要安装cp2102驱动程序，使用一根Mini USB数据线连接电脑。这里就详细介绍使用USB固件烧写流程。

首先从crazepony的百度云盘中下载开发工具，点击这里。解压压缩文件得到3个文件夹。

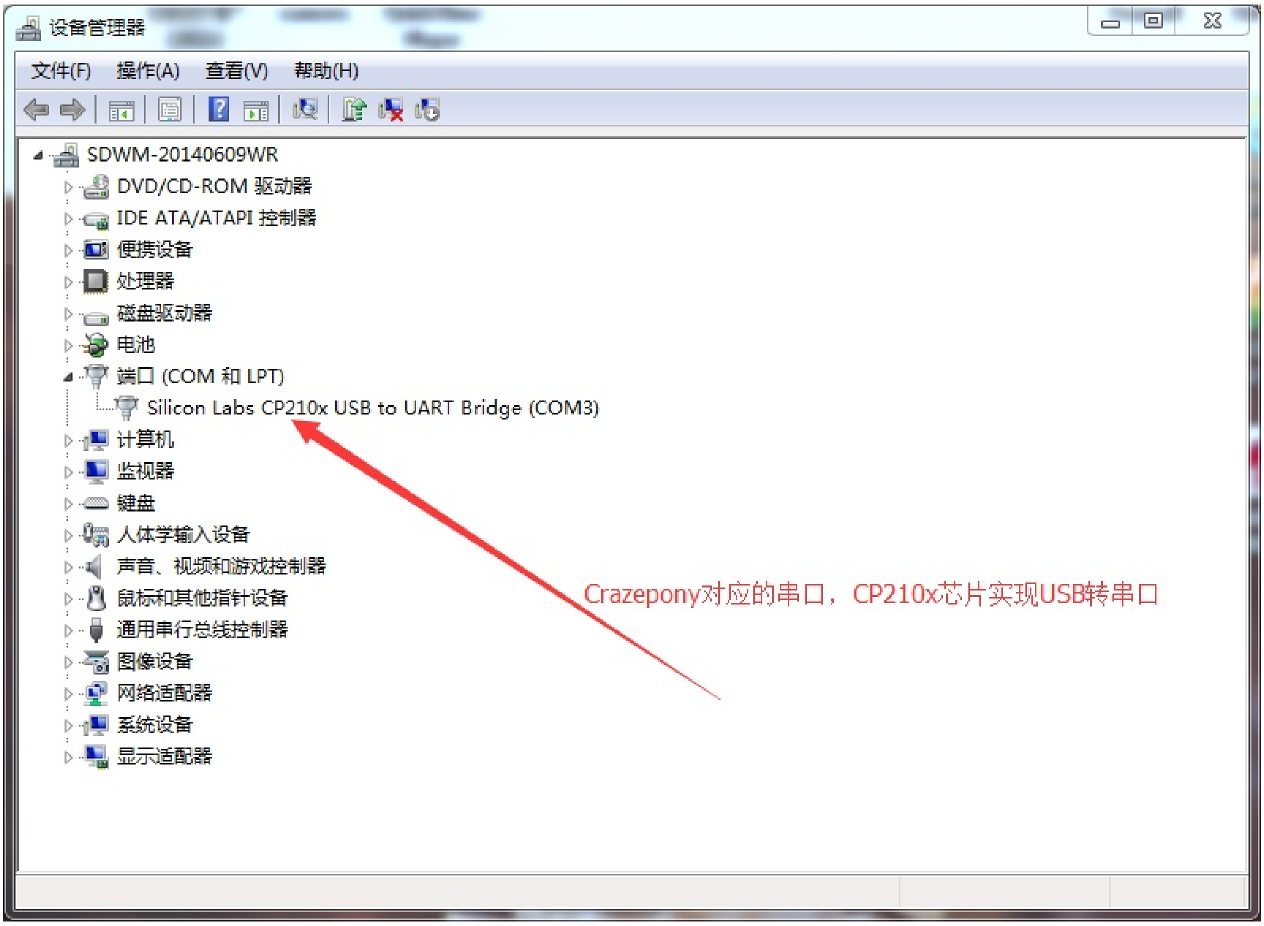
安装cp2102驱动

cp2102是Crazepony上使用的USB转串口芯片。cp2102和STM32芯片以串口相连，和电脑PC以USB接口相连，这是Crazepony能够接上USB线对飞控/遥控器进行固件烧写编程和调试信息打印的原因。

首先要把cp2102的驱动在电脑PC上装好，这样电脑作为Host才能够识别到cp2102。这个驱动在32位windows xp系统/64位 windows 7系统下都测试通过。

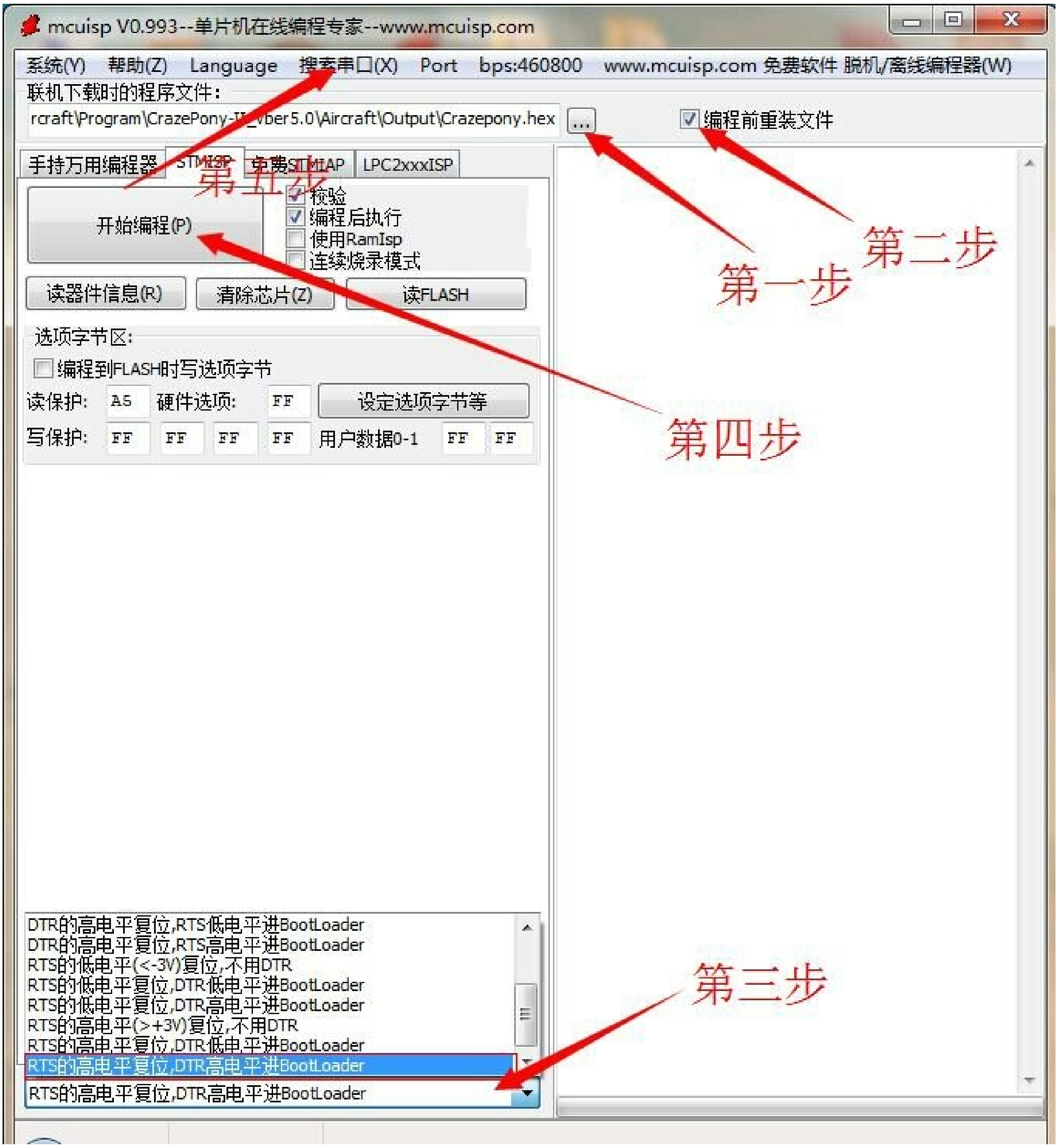


安装成功之后，在我的电脑->设备管理->端口（COM和LPT），就可以看到Crazepony对应的串口号，例如下图为COM3。看到该串口中有关于CP210x的描述。



烧入hex文件

打开ISP下载器，载入需要下载的hex文件，具体如下图所示。



在硬件设计上，直接使用CP2102复位STM32并且引导进入串口升级固件的ISP下载模式。CP2102使用RTS高电平复位STM32芯片，然后使用DTR将STM32的Boot0引脚拉低，STM32进入串口升级固件的ISP下载模式。所以在ISP下载器上，必须选择左下角的“RTS的高电平复位，DTR高电平进Bootloader”。

查看打印信息

连上USB线，打开串口助手，波特率设置为115200，查看串口打印信息。默认每隔1秒打印一次传感器数据信息。

> Crazepony默认出厂固件串口打印信息和上位机数据都关闭。USB串口打印和USB上位机同时只能够使用一个，不能够同时使用。原因是串口上的上位机数据在串口助手上看到的是乱码。要使用串口打印信息，请开启 `SysConfig.h` 中的 `DEBUG_UART` 宏。要使用上位机查看信息，请关闭 `SysConfig.h` 中的 `DEBUG_UART` 宏，并开启 `main.c` 中的 `CommPCUploadHandle()` 函数。



J-Link的使用及常见问题

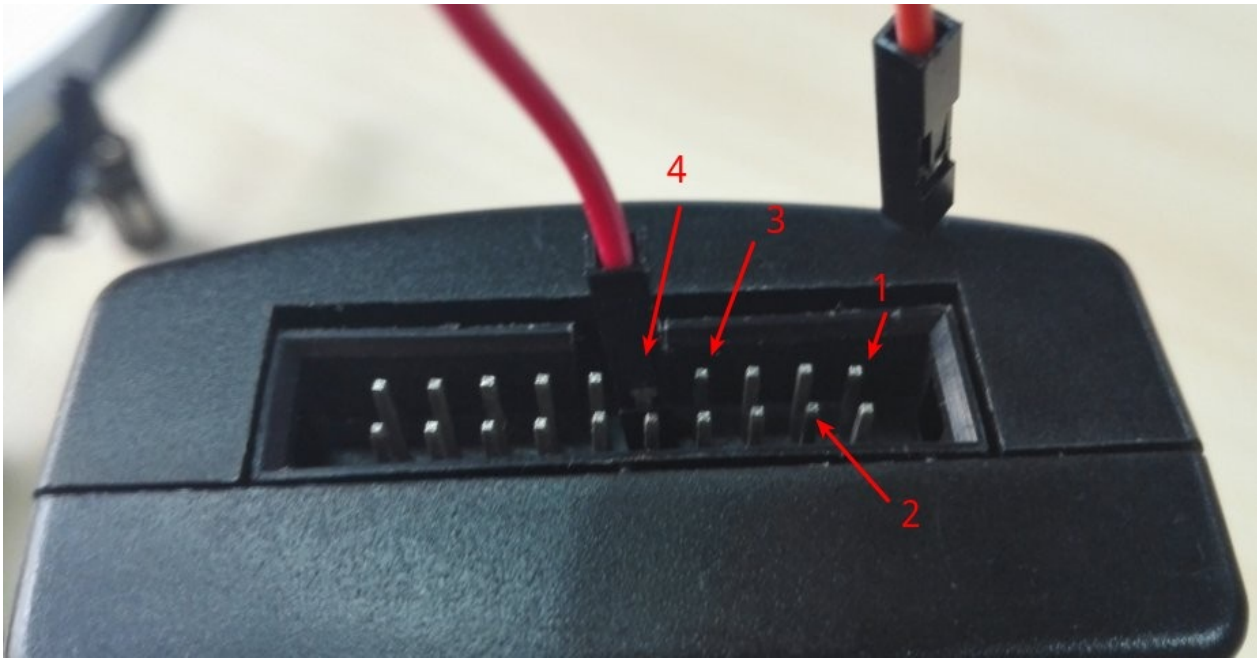
> 作者：nieyong

crazepony除了可以使用usb接口下载程序之外，还将swd接口预留到了板外。可以使用J-Link，ST-Link等工具进行代码的烧写/在线调试等。下面介绍J-Link的配置和常见问题。

J-Link与Crazepony的接线

一般我们买到的J-Link如下所示，是一个20针的排线口。对于SWD接口，我们只需要使用3根线就可以。J-Link和Crazepony的连接如图如下，使用彩虹排线连接就可以。（Crazepony的SWD接口排阵默认未焊接）

- 1所指排针与飞控的VCC相连（如果飞机自己供电，可以不连接）
- 2所指排阵与飞控的GND相连
- 3所指排阵并飞控的DIO相连
- 4所指排阵并与飞控的CLK相连

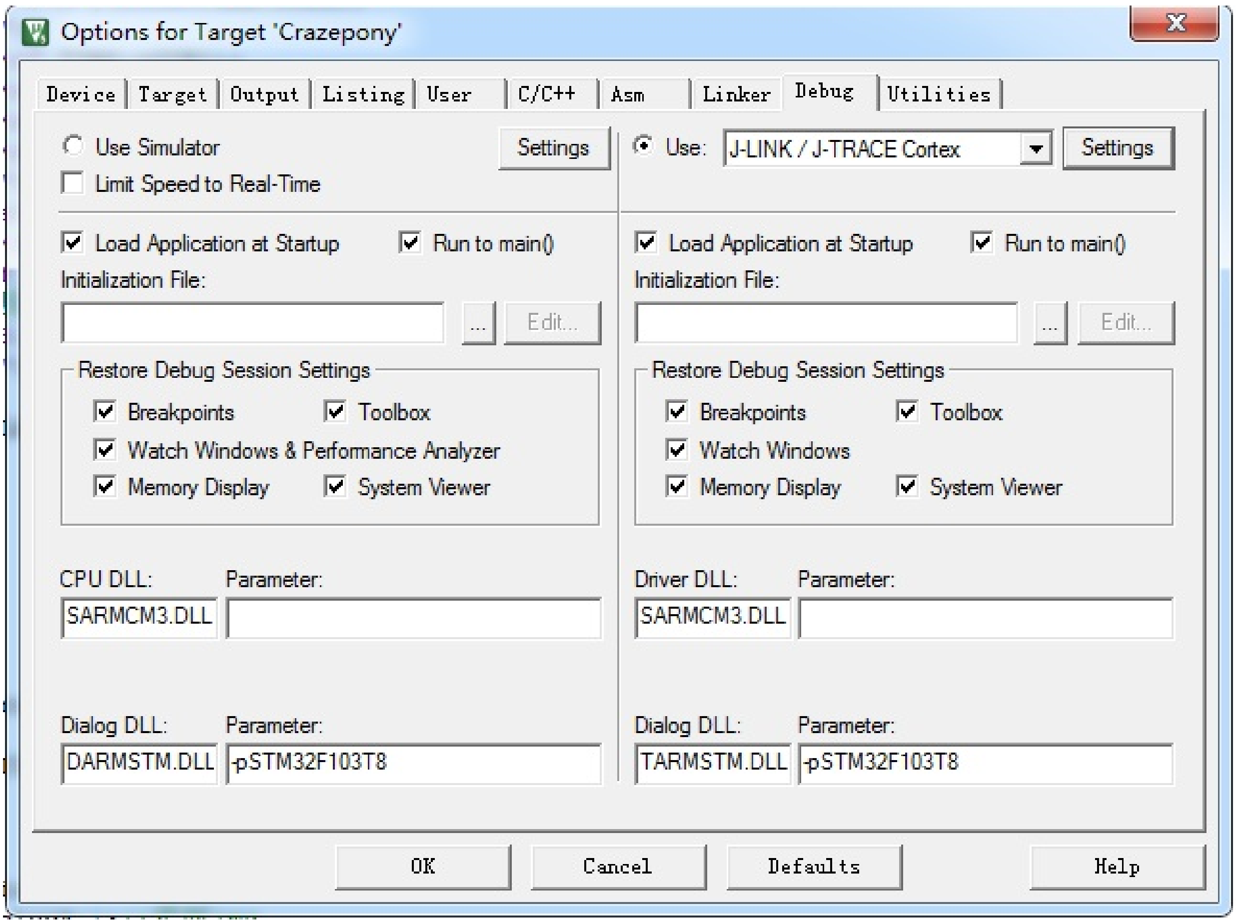


J-Link与Crazepony相连示意图。Crazepony遥控器的连接也是同理。

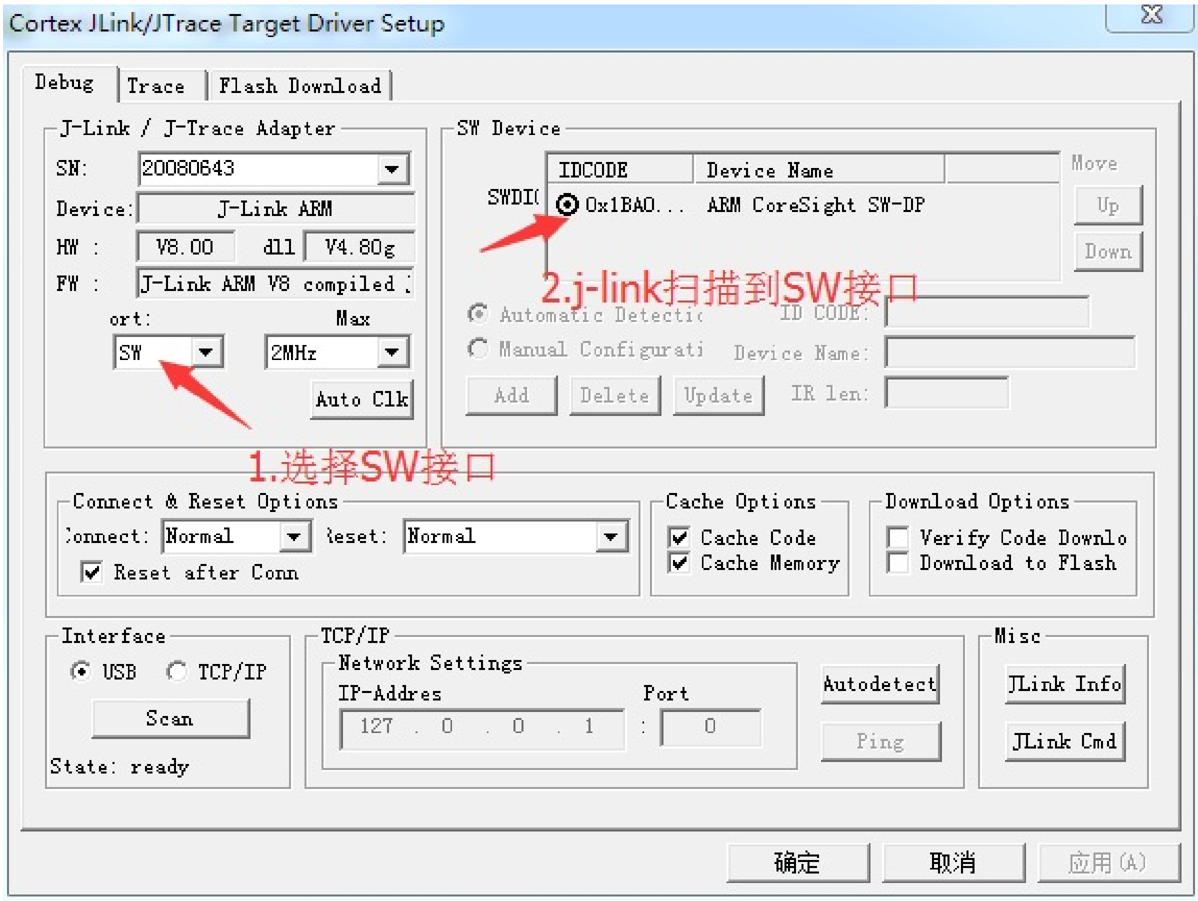


J-Link的配置和使用

使用keil打开crazepony工程文件，点击Options for Target 'Crazepony'，选择Debug部分，下拉选择J-Link/J-Trace Cortex.



然后点击设置，配置如下。



在选择了SW接口之后，将飞控开关打到on，点击Auto Alk，如果连接正确，就会扫描到开发板上的SW接口。

J-Link固件烧写

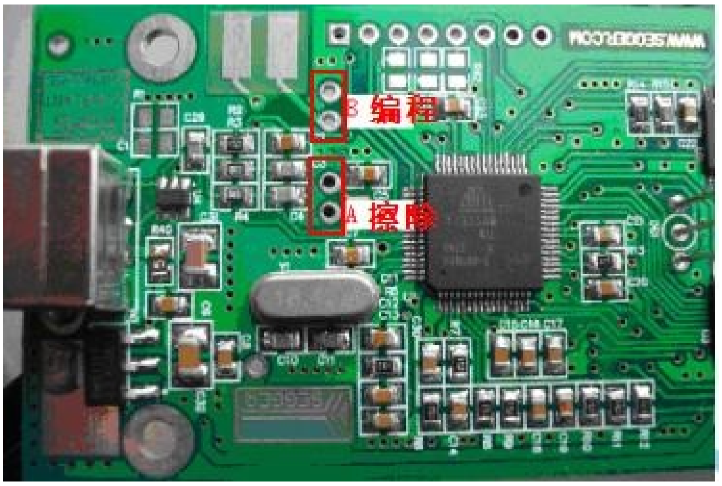
在插拔的过程中，J-Link经常会出现固件丢失无法使用的情况，这时需要重刷J-Link调试器的固件。或者从keil 4切换导Keil 5的时候，需要升级J-Link调试器固件。

下面以J-Link V8型号调试其为例说明如何重刷J-Link的固件。

安装AT91-ISP，安装完成后在桌面上将有两图标。



擦除芯片，进入编程模式。J-Link面板上有两对接口，其中A为擦除，B为编程接口。



擦除芯片过程，将J-Link连接到电脑，短接A两孔约5秒钟，断开，然后拔掉J-Link与电脑的连接。如果重新连接电脑，led灭掉，在设备管理中找不到任何相关设备，则表示擦除成功。

擦除成功后，要进入编程模式。进入编程模式的过程，短接B两孔，连接接J-Link到电脑，至少约10秒钟，拔掉USB线，断开B处短接。再次插上电脑，在设备管理中有未知设备出现，则表示进入编程模式成功。



如果已经安装了AT91-ISP，并且其驱动运转正常，那么可以J-Link被识别为一个USB转串口设备，表示进入编程模式成功。我们测试有些电脑安装之后驱动会不正确，请换一台电脑重试。只有驱动正常，入下图所示，才可以烧入固件。



双击SAM-PROG v2.4，运行烧录软件如下。



将J-Link通过USB线与PC机连接。此时，SAM-PROG v2.4软件中的Write Flash 按钮将变为有效。注意，先打开SAM-PROG v2.4，再连接J-LINK与PC机。

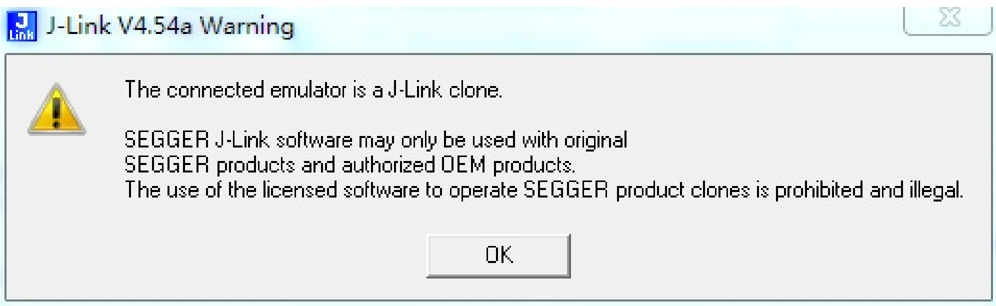
点击Write Flash按钮，烧录固件，待烧录完成后，Active Connection：将变为 1。

如果遇到问题是write flash一直都是灰色的，那么可能是AT91-ISP驱动未正常运行，无法J-Link识别为USB转串口设备。换台电脑就可以。

至此,J-LINK的固件已经更新完毕，正常情况下，连接电脑与J-LINK时，JLINK的指示灯将闪烁。如果J-Link驱动安装正确，那么LED最后常绿。

Keil 5下面J-Link失效的问题

在Keil 5下面，老版本的J-Link可能无法使用，出现下面的提示。这个时候需要升级J-Link固件。参考文档MDK5安装与JLINK问题解决方法(支持代码自动补全)

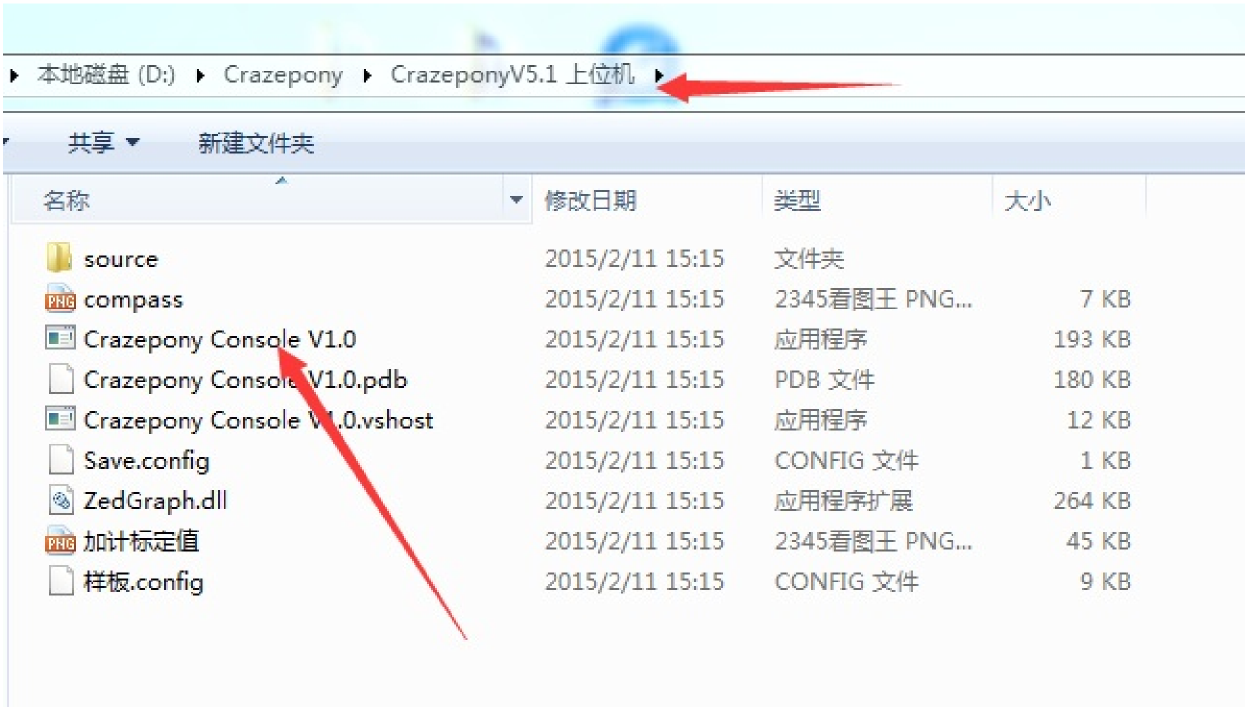


Crazepony上位机使用手册

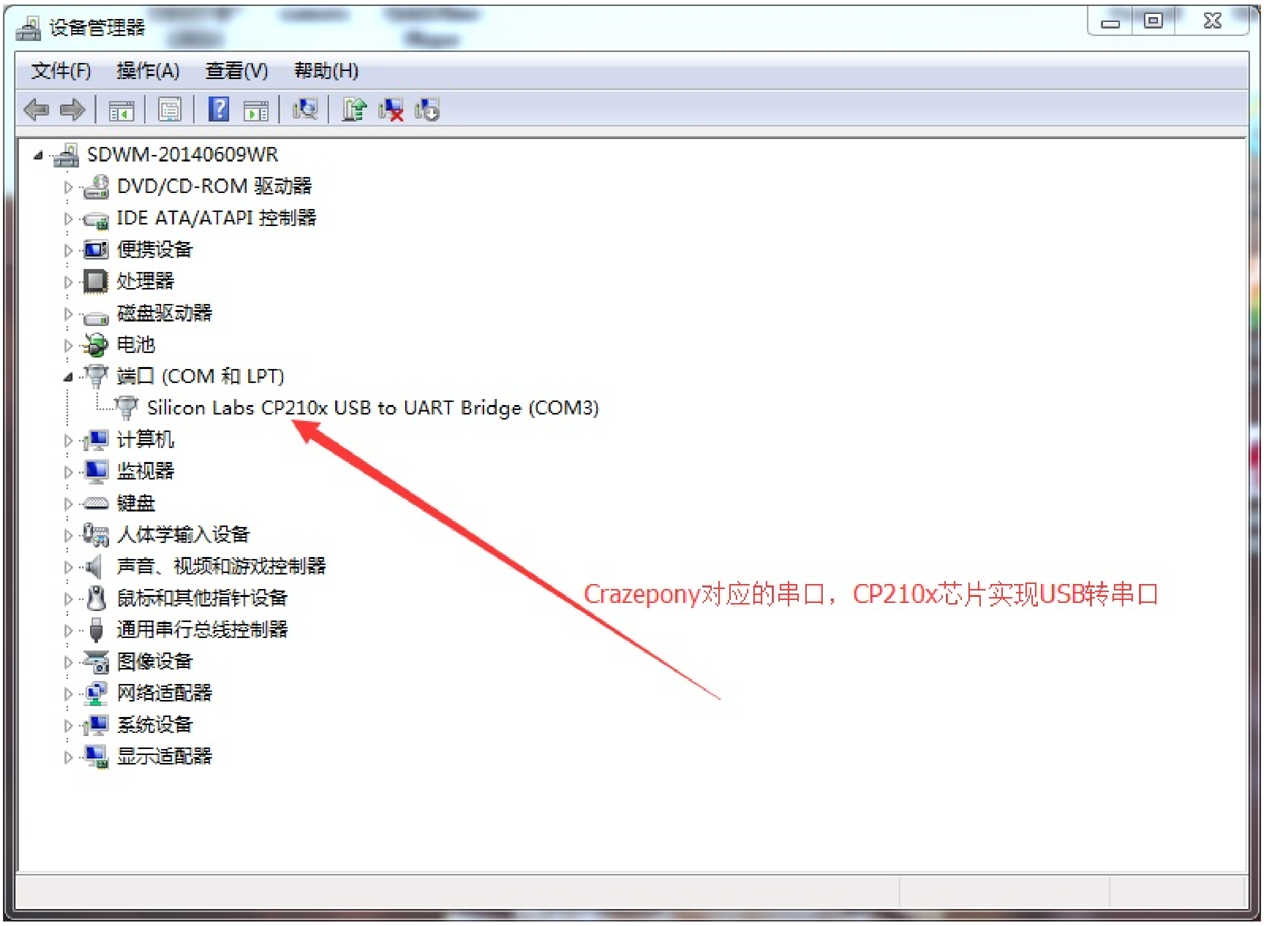
> 作者：nieyong

Crazepony四轴飞行器提供一个Windows PC端上位机软件，用于查看姿态数据，调节PID等操作。该上位机使用C#编写，源代码暂时没有开源。下面介绍上位机的配置和使用。

首先，从百度云盘下载最新的上位机，得到的文件如下，点击其中的可执行文件，打开上位机软件。



将Crazepony四轴使用usb线连接到电脑，并且安装usb转串口驱动（关于如何安装该驱动，参考《固件烧写》中的安装cp2102驱动部分）。我的电脑->设备管理->端口（COM和LPT），就可以看到Crazepony对应的串口号，例如下图为COM3。Crazepony是使用CP210x芯片实现的USB转串口功能，所以会看到该串口中有关于CP210x的描述。



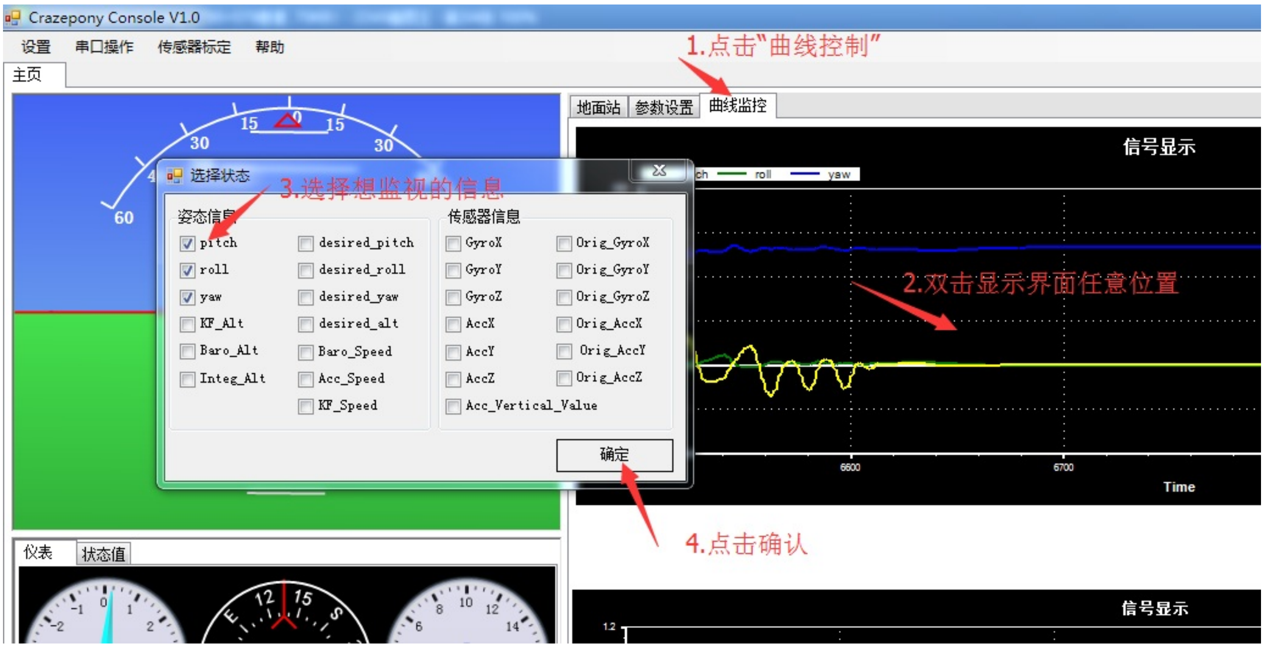
接下来是在上位机中选择Crazepony串口。步骤如下图所示。如果提示串口已经被打开，可以在左上角的串口操作中选择关闭串口，再进行操作。如果串口正常连接并且打开，那么上位机就可以接收Crazepony的数据，晃动一下Crazepony会发现左上角的视图在变化。



查看/修改PID参数。点击右边的参数设置，可以配置Crazepony的PID参数。

todo

监视姿态信息。电机右边的曲线控制，可以监视Crazepony的实时姿态信息。



蓝牙无线调参

除了使用usb线连接上位机之外，还可以使用蓝牙无线连接上位机，这样可以省去连接线的束缚。

- 需要电脑有蓝牙设备
- 开飞机电源，打开电脑蓝牙设备查找，找到一个叫crazepony的设备配对连接，配对码为1234
- 打开电脑设备管理器，查看蓝牙透传串口号
- 打开上位机，选择刚刚看到的串口号，波特率115200，确认连接。

自主悬停&高度融合

> 作者：nieyong

自主悬停是Crazepony玩家问得最多的问题，也是技术难度最高的问题。

首先我们明确什么是自主悬停？下面是一位资深航模玩家对于自主悬停的解释。

> 飞行器能够悬停在某个位置，在发生了偏离之后能够自动校正并且回到原来悬停的位置，并且能够在该位置保持足够长的时间，至少2分钟以上，那么我们认为该飞行器是能够自主悬停的。

飞行器在空中是一个三维坐标(x,y,z)位置，涉及到两个维度的悬停。第一是在水平面上(x,y)，飞行器不能够左右或者前后漂移。第二是在垂直方向上（又叫Z轴维度），飞行器不能够发生掉高等现象。针对这两个不同的维度，有不同的技术进行解决。

在水平面上，为了确定飞行器的位置，一般使用GPS进行定位，室内一般使用光流进行定位（室内没有GPS信号）。

在垂直方向上，一般使用气压计进行定高，室内等条件下可以使用超声波模块。

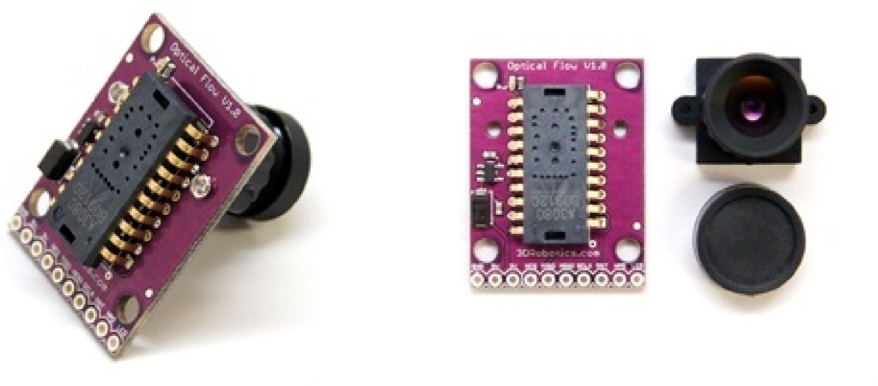
能否只用6维数据实现悬停

能否只使用3维加速度和3维陀螺仪这6个维度的数据实现自主悬停？而不再使用气压计和GPS等。现在就我了解，是很难实现。因为只使用这个6维数据，无法知道飞行器在空中的绝对坐标，只能够由6维数据获得相对偏移。没有绝对坐标，就无法对位置实现闭环控制。

当然，我们看到很多只是用了MPU6050的玩具四轴，也能够实现一定程度的悬停，但是那并不是真正意义上的自主悬停。

水平面的定位

室外使用GPS进行水平定位，室内一般使用光流技术（optical flow）。光流传感器其实就是一个摄像头，安装在飞行器上垂直拍摄地面。光流技术就是要对拍摄到的画面进行视觉处理，以此知道相对于地面坐标位置。APM飞控中关于光流传感器的介绍。



垂直方向定位

在垂直方向上，使用高精度气压计（如MS5611）进行定位。在飞行器处于较低高度时，可以使用超声波进行高度定位。

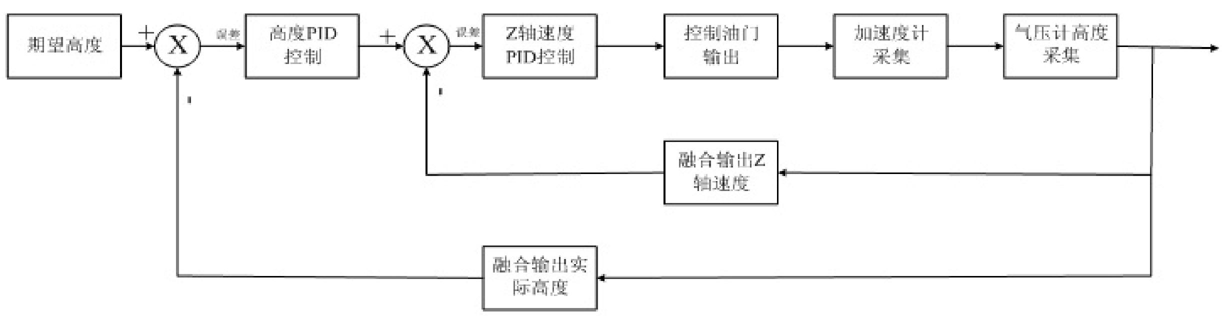
Crazepony的自动悬停

飞行器要想实现真正意义的自主悬停需要有自稳和空中绝对坐标。Crazepony有一个高精度气压计MS5611，所以可以实现Z轴的自主悬停。在水平方向上，crazepony没有绝对坐标，无法实现悬停。

在Z轴自主悬停上，Crazepony有一个高精度气压计MS5611，通过气压传感器采集的大气压值计算出来，将气压传感器采集值进行校正后，再通过温度二阶补偿，得到准确的大气压值，最后经过气压转换高度公式就可以得到一个相对于起飞地点的绝对高度（参考气压计MS5611）。

MS5611气压计的精度为10CM，因此需要融合加速度计互补滤波得到比较合适的高度值，Z轴速度值和加速度值。用高度作外环，速度作内环形成高度双环PID控制器，调节输出油门实现Z轴的自主悬停。高度融合在文件 `Altitude.c` 中实现，高度双环PID控制在 `Control.c` 文件中的 `CtrlAlti()` 函数中实现。

高度双环PID控制总体流程。



代码实现在 `Control.c` 文件中的 `CtrlAlti()` 函数中。

```
……
// pid and feedforward control . in ned frame
posZErr= -(altSp - alt);
posZVelSp = posZErr * alt_PID.P + spZMoveRate * ALT_FEED_FORWARD;
//consider landing mode
if(altCtrlMode==LANDING)
    posZVelSp = LAND_SPEED;

velZ=nav.vz;
velZErr = posZVelSp - velZ;
valZErrD = (spZMoveRate - velZ) * alt_PID.P - (velZ - velZPrev) / dt;    //spZMoveRate is
velZPrev=velZ;

thrustZSp= velZErr * alt_vel_PID.P + valZErrD * alt_vel_PID.D + thrustZInt;    //in ned f
……
```

在水平方向上，由于Crazepony没有绝对坐标，使用3维加速度和3维陀螺仪这6个维度的数据进行双环PID控制形成一个自稳系统，可实现短时间定点脱控悬停，但是受误差影响产生偏离，没有水平绝对坐标就无法回到原来的位置，所以Crazepony是无法实现正真意义上自动悬停的。

Z轴悬停-悬停油门基准值

在Z轴悬停代码中，定义了一个悬停油门基准值 `HOVER_THRU` ，该值决定着飞行器脱控悬停的时候需要的油门大小。对悬停起到至关重要的作用。 在版本本9bf1e83f之后，该宏定义由函数 `estimateHoverThru()` 代替，根据当前电池电压大小，输出对应的油门悬停基准值，能够彻底解决充满电之后第一次飞行直接往上冲的问题。

油门基准值的选取对悬停是至关重要的。例如，我们换了一批扭力更大的电机，那么在飞行器重量不变的情况下，使用原来的基准值就会出现脱控之后一直往上飞的情况。这个时候，我们就需要将该值改小，以此适应这批次扭力大的电机。

悬停油门基准值将会直接影响Z轴悬停，选取不当或者系统参数发生变化时，就会出现往上飞或者掉高的问题。这里的系统参数发生变化，包括电池电力变化，飞机重量改变，电机扭力变化等。如果使用固定值，在电池充满电之后，会出现起飞之后往上冲的问题。所以在版本9bf1e83f之后，该悬停油门基准值由函数 `estimateHoverThru()` 输出。根据电池电压不同，使用不同的油门基准值。

```
float estimateHoverThru(void){
    float hoverHru = 0.55f;

    //电池电压检测
    Battery.BatteryAD  = GetBatteryAD();
    Battery.BatteryVal = Battery.Bat_K * (Battery.BatteryAD/4096.0) * Battery.ADRef;//实际

    if(Battery.BatteryVal > 4.05){
        hoverHru = -0.35f;
    }else if(Battery.BatteryVal > 3.90){
        hoverHru = -0.40f;
    }else if(Battery.BatteryVal > 3.80){
        hoverHru = -0.45f;
    }else if(Battery.BatteryVal > 3.70){
        hoverHru = -0.50f;
    }else{
        hoverHru = -0.55f;
    }

    return hoverHru;
}
```

Z轴悬停-最小油门值

根据机重和动力系统的输出，我们还为Crazepony设定了一个最小的油门输出值 `THR_MIN`，这样能够限制最大的下降速度。可以避免下降速度过快，机身失去平衡。

限定最小的油门输出值 `THR_MIN` 本来是为了解决下降过快的问题，有时候也会带来副作用。例如充电后第一次飞行，由于动力输出强劲，如果这个最小油门输出值设置得过大，那么即使将摇杆油门拉到了最低，飞机也会一直往上冲。在版本7ee226a之后，该宏被修改为函数 `estimateMinThru()`，将输出和当前电池电压挂钩，能够解决往上冲下拉油门时无法受控的问题。

```
float estimateMinThru(void){
    float minThru = -0.55f;

    //电池电压检测
    Battery.BatteryAD  = GetBatteryAD();
    Battery.BatteryVal = Battery.Bat_K * (Battery.BatteryAD/4096.0) * Battery.ADRef;//实际

    if(Battery.BatteryVal > 4.05){
        minThru = -0.30f;
    }else if(Battery.BatteryVal > 3.90){
        minThru = -0.40f;
    }else{
        minThru = -0.55f;
    }

    return minThru;
}
```

2.4G天线设计及遥控距离

> 作者：nieyong

2.4G天线设计

2.4G天线设计，直接影响到遥控的距离。我们现在使用了陶瓷天线，尽量减少天线所占面积。

带PA的发射模块

遥控器上使用带有PA的发射模块，可以将2.4G遥控的距离扩大到100米左右。

摄像头和图传

> 作者：nieyong

很多小伙伴都有在微型四轴上悬挂摄像头或者图传模块的需求。Crazepony默认是不带摄像头的。下面我们整理了一些微型摄像头图传模块，方便悬挂到小四轴上。

Crazepony默认的载重在**20g**左右。如果换成**820**电机载重可以提升。

5.8G图传摄像头一体化模块

该模块带有5.8G图传和微型摄像头，只有6g左右，可以悬挂到Crazepony小四轴下面。该模块可以从淘宝大力模型店铺购买到，如下所示。

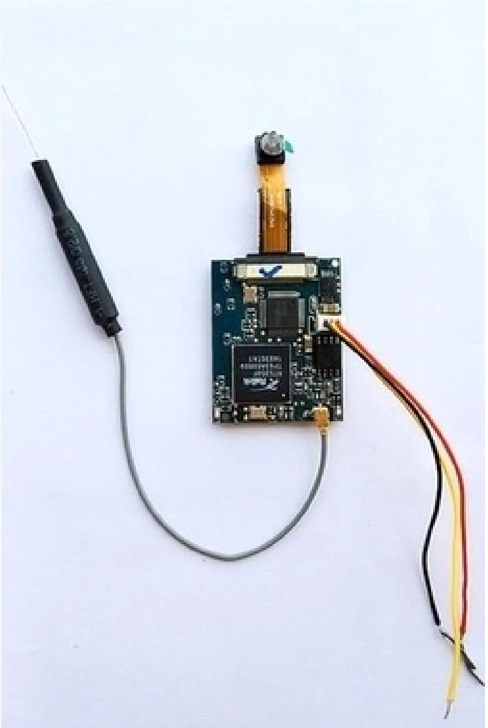


5.8G模块需要配备相应的5.8G接收机才能够将图传数据接收。

wifi图传摄像头一体化模块

该模块带有wifi图传和微型摄像头，只有12g。配上相应的手机APP就能够实时图传。

淘宝购买地址。



摄像头录像模块

如果不想要图传，只需要挂载摄像头做录像拍照，我们可以使用单一的摄像头模块。

我们测试过Crazepony可以携带10g左右的微型摄像头。该摄像头只有大拇指大小，可以插入SD卡保存录像图像。该摄像头只需要从Crazepony机身板上供电即可。

Crazepony通信协议

> 作者：nieyong

2.4G遥控器NRF24L01

Crazepony机身上2.4G射频芯片NRF24L01和主控STM32使用SPI总线连接在一起。

Crazepony遥控器上，我们使用了市面上标准排插接口的NRF24L01模块。

手机APP通信协议

Crazepony APP和飞控之间通信协议使用了MWC飞控协议（MSP，Multiwii Serial Protocol），详见MSP协议格式。

MWC协议规定了飞控和上位机（或者手机APP）信息交流的基本格式。

MWC具体实现，可以查看Crazepony的Android APP源代码中的 `Protocol.java` 文件。

四轴飞行器平衡稳定调试

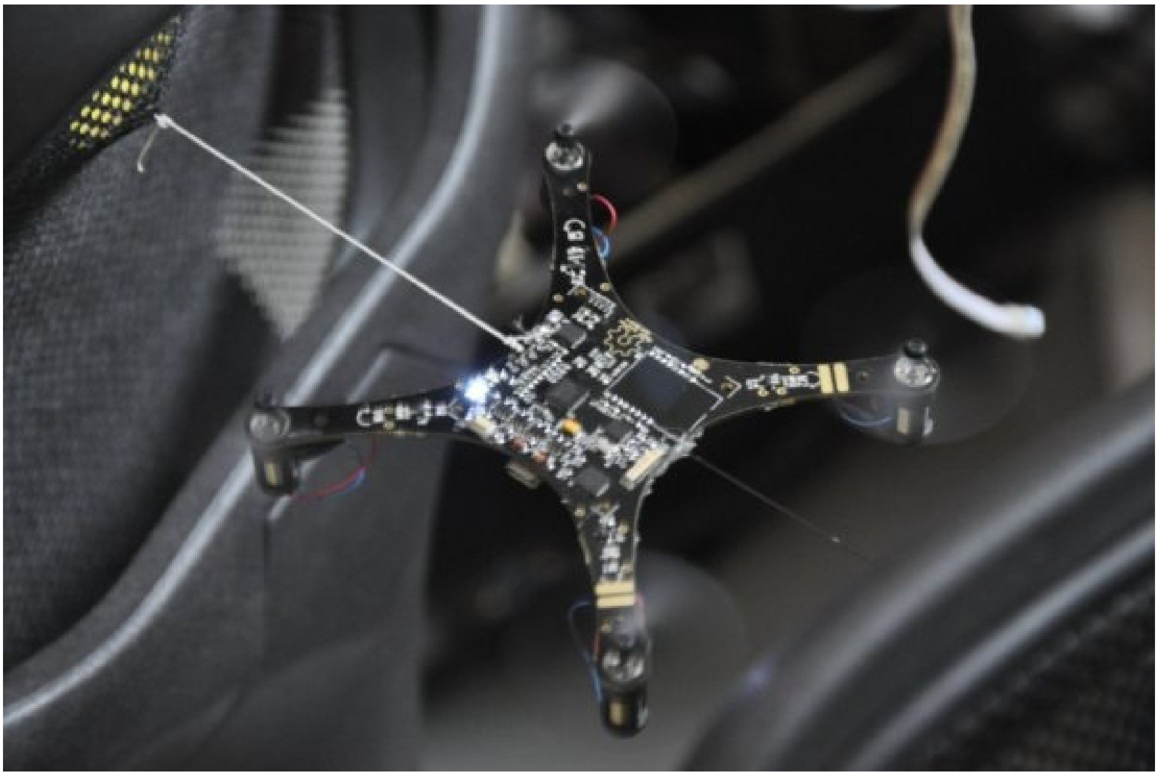
> 作者：nieyong

下面介绍Crazepony四轴飞行器平衡稳定的调试方式，这些调试的方式同时也可以用于其它四轴的调试。

固定调试

可以将飞行器用绳子栓住，然后调试最基本的平衡和稳定。这样一般可以获得一个大概的PID值。只有这步调试基本稳定，才会进行后面更加精细深入的调试。

栓住飞行器的办法有很多，例如用两根绳子拴在两个臂膀上，或者用一根绳子将飞行器底部拴住。下面是Crazepony早期平衡调试的照片。



无线调试

使用蓝牙模块，在飞行器飞行的过程中实时将希望观察的数据传回上位机。

Crazepony动力效率和续航时间

> 作者：nieyong

很多人会问，充一次电，*Crazepony*能够飞多久呢？才*6*分钟，要是能够飞上*10*分钟就好了。为什么Crazepony只能够飞6分钟，如何才能够提高飞机的动力效率，提升飞行时间？

那么这里我们分享一些我们调试实验的结果和经验，对大家有所帮助。

电池容量

Crazepony的电池容量，在5.2版本由原来的380mAh换成了650mAh，能够提供更多的动力能源。

当然，并不是一味的增加电池容量就能够增加续航能力。更大的电池容量需要有更重更大的电池，本身就增加了飞机的负重，所以电池容量一定要在合理的范围内。

电机&桨叶的选型

我们测试发现，电机和桨叶的选型对续航能力的提升最为显著。

下面是同样的650mAh电池充满电的情况下，使用两种不同桨叶的续航能力对比结果。以下测试在720电机下进行。



使用Crazepony原配75mm桨叶，电机发热发烫明显，飞行时长6min。



使用parrot的mini drone原配66cm桨叶，电机明显发热减少，飞行时长7min。

为什么原配的75mm桨叶动力效率会低很多呢？我们仔细分析后发现，75mm桨叶本来就比66cm桨叶长，而且叶面受力面积大，最重要的是螺距要大，所有的这些都要求有强劲动力的电机和它相匹配（也就是小KV值的电机）。而我们的720空心杯单机本来就是高KV值电机，和该桨叶的匹配并不是非常理想。

820空心杯电机

现在我们使用的720空心杯电机，在连续飞行2min之后，就会发烫厉害。这主要是由于720电机的动力有限，带动46g的Crazepony比较吃力。

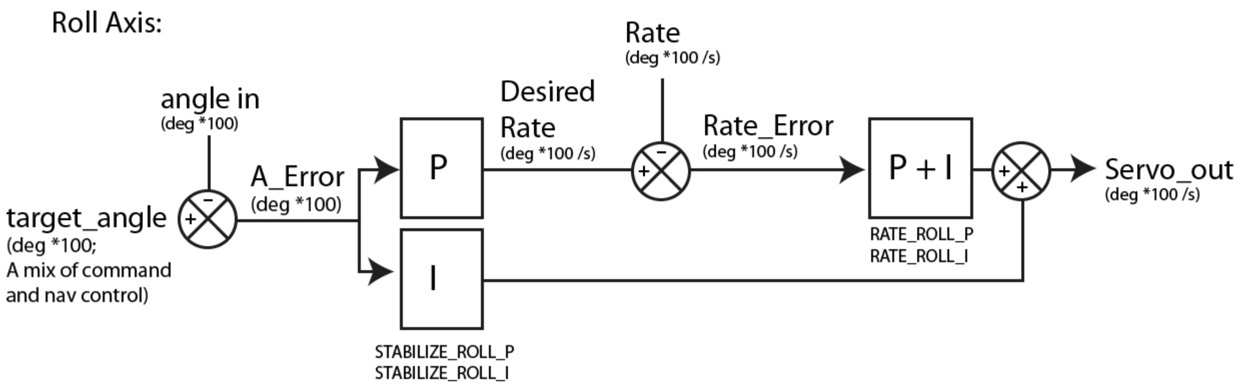
我们有尝试使用820空心杯电机，该电机直径为8mm，动力非常强劲，续航能力更强。在解决了电机装配的问题之后，我们会默认使用该型号电机。


820电机，直径为8mm，相对于原来的
720电机，动力强劲，续航增加

四轴飞行器算法讲解

本章主要是Crazepony核心的算法讲解。其中最重要的就是姿态解算部分内容。

姿态解算是指把陀螺仪、加速度计、罗盘等数据融合在一起，得出飞行器的空中姿态，也叫做姿态融合。姿态解算的过程，涉及到传感器数据读取与滤波，四元数与旋转，姿态解算框架，长期融合/快速融合。



姿态解算

> 作者：nieyong

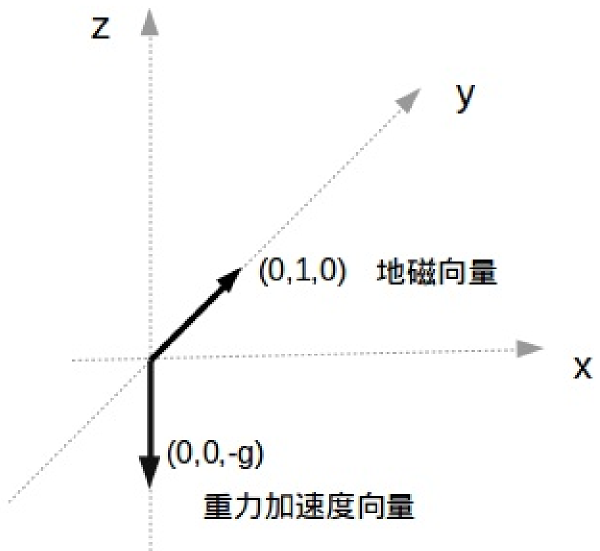
姿态解算的英文是attitude algorithm，也叫做姿态分析，姿态估计，姿态融合。姿态解算是指把陀螺仪、加速度计、罗盘等的数据融合在一起，得出飞行器的空中姿态。

飞行器从陀螺仪的三轴角速度通过四元数法得到的俯仰、偏航和滚转角，这是快速解算，结合三轴地磁和三轴加速度得到漂移补偿和深度解算。

数学模型

姿态是用来描述一个刚体的固连坐标系和参考坐标系之间的角位置关系，有一些数学表示方法。很常见的就是欧拉角，四元数，矩阵，轴角。

姿态解算需要解决的是四轴飞行器和地球的相对姿态问题。地球的坐标系又叫做地理坐标系，是固定不变的。正北，正东，正向上构成了这个坐标系的X，Y，Z轴，我们用坐标系R表示。四轴飞行器上固定着一个坐标系，用坐标系r表示。那么我们就可以用欧拉角，四元数等来描述r和R的角位置关系。这就是四轴飞行器姿态解算的数学模型和基础。



地球坐标系 R（地理坐标系）

角位置关系测量

如上所说，地球坐标系R是固定的。四轴飞行器上固定一个坐标系r，这个坐标系r在坐标系R中运动。那么如何知道坐标系r和坐标系R的角位置关系呢，也就是怎么知道飞行器相对于地球这个固定坐标系R转动了一下航向，或者侧翻了一下机身，或者掉头下栽。这就是传感器需要测量的数据，传感器包括陀螺仪，加速度计，磁力计。通过获得这些测量数据，得到坐标系r和坐标系R的角位置关系，这就是姿态解算。

姿态的数学表示方法

> 作者：nieyong

姿态有多种数学表示方式，常见的是四元数，欧拉角，矩阵和轴角。他们各自有其自身的优点，在不同的领域使用不同的表示方式。在四轴飞行器中使用到了四元数和欧拉角。

四元数

四元数是由爱尔兰数学家威廉·卢云·哈密顿在1843年发现的数学概念。从明确地角度而言，四元数是复数的不可交换延伸。如把四元数的集合考虑成多维实数空间的话，四元数就代表着一个四维空间，相对于复数为二维空间。

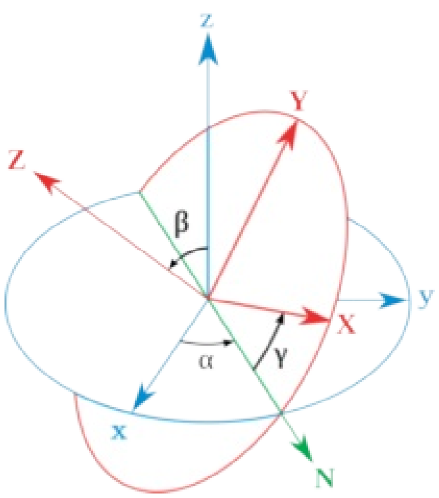
四元数大量用于电脑绘图（及相关的图像分析）上表示三维物件的旋转及方位。四元数亦见于控制论、信号处理、姿态控制、物理和轨道力学，都是用来表示旋转和方位。

相对于另几种旋转表示法（矩阵，欧拉角，轴角），四元数具有某些方面的优势，如速度更快、提供平滑插值、有效避免万向锁问题、存储空间较小等等。

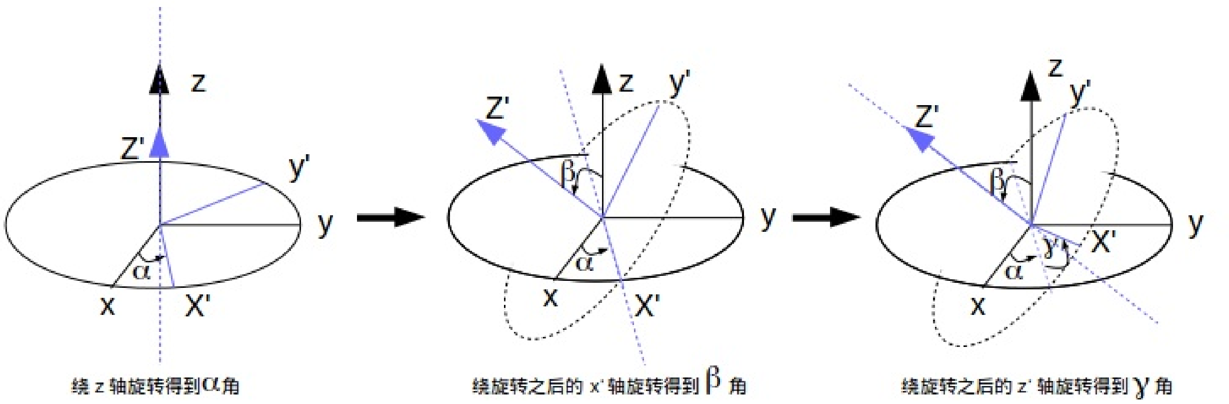
以上部分摘自维基百科-四元数。

欧拉角

莱昂哈德·欧拉用欧拉角来描述刚体在三维欧几里得空间的取向。对于在三维空间里的一个参考系，任何坐标系的取向，都可以用三个欧拉角来表现。参考系又称为实验室参考系，是静止不动的。而坐标系则固定于刚体，随着刚体的旋转而旋转。



以上部分摘自维基百科-欧拉角。下面我们通过图例来看看欧拉角是如何产生的，并且分别对应哪个角度。



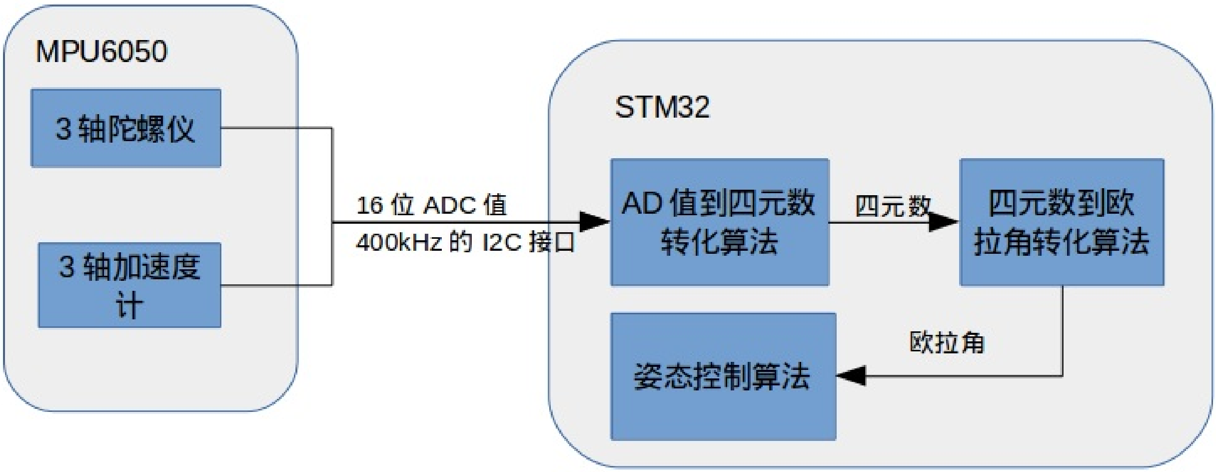
姿态解算为什么要用四元数和欧拉角

姿态解算的核心在于旋转，一般旋转有4种表示方式：矩阵表示、欧拉角表示、轴角表示和四元数表示。矩阵表示适合变换向量，欧拉角最直观，轴角表示则适合几何推导，而在组合旋转方面，四元数表示最佳。因为姿态解算需要频繁组合旋转和用旋转变换向量，所以采用四元数保存组合姿态、辅以矩阵来变换向量的方案。

总结来说，在crazepony中，姿态解算中使用四元数来保存飞行器的姿态，包括旋转和方位。在获得四元数之后，会将其转化为欧拉角，然后输入到姿态控制算法中。

姿态控制算法的输入参数必须要是欧拉角。AD值是指MPU6050的陀螺仪和加速度值，3个维度的陀螺仪值和3个维度的加速度值，每个值为16位精度。AD值必须先转化为四元数，然后通过四元数转化为欧拉角。这个四元数可能是软解，主控芯片（STM32）读取到AD值，用软件从AD值算得，也可能是通过MPU6050中的DMP硬解，主控芯片（STM32）直接读取到四元数。具体参考《MPU60x0的四元数生成方式介绍》。

下面就是四元数软解过程，可以由下面这个框图表示：



扩展阅读-四元数的运算

下面介绍一下四元数，然后给出几种旋转表示的转换。这些运算在crazepony的代码中都会遇到。

四元数可以理解为一个实数和一个向量的组合，也可以理解为四维的向量。这里用一个圈表示q是一个四元数（很可能不是规范的表示方式）。

$$\mathring{q}=\{w,\vec{v}\}=[w \quad x \quad y \quad z]^T$$

四元数的长度（模）与普通向量相似。

$$|\mathring{q}|=\sqrt{w^2+x^2+y^2+z^2}$$

下面是对四元数的单位化，单位化的四元数可以表示一个旋转。

$$\hat{\mathring{q}}=\frac{\mathring{q}}{|\mathring{q}|}=\begin{bmatrix}\frac{w}{|\mathring{q}|} & \frac{x}{|\mathring{q}|} & \frac{y}{|\mathring{q}|} & \frac{z}{|\mathring{q}|}\end{bmatrix}^T$$

四元数相乘，旋转的组合就靠它了。

$$\mathring{q}_0=\mathring{q}_1\cdot\mathring{q}_2$$

$$\begin{cases} w_0=w_1w_2-x_1x_2-y_1y_2-z_1z_2 \\ x_0=w_1x_2+x_1w_2+y_1z_2-z_1y_2 \\ y_0=w_1y_2-x_1z_2+y_1w_2+z_1x_2 \\ z_0=w_1z_2+x_1y_2-y_1x_2+z_1w_2 \end{cases}$$

旋转的“轴角表示”转“四元数表示”。这里创造一个运算q(w,θ)，用于把绕单位向量w转θ角的旋转表示为四元数。

$$\{\hat{w},\theta\}\rightarrow\mathring{q}(\hat{w},\theta)=\{\cos(\frac{\theta}{2}),\hat{w}\cdot\sin(\frac{\theta}{2})\}$$

通过q(w,θ)，引伸出一个更方便的运算q(f,t)。有时需要把向量f的方向转到向量t的方向，这个运算就是生成表示对应旋转的四元数的（后面会用到）。

$$\mathring{q}(\vec{f},\vec{t})=\mathring{q}(\frac{\vec{f}\times\vec{t}}{|\vec{f}\times\vec{t}|},\operatorname{atan2}(|\vec{f}\times\vec{t}|,\vec{f}\cdot\vec{t}))$$

然后是“四元数表示”转“矩阵表示”。再次创造运算，用R(q)表示四元数q对应的矩阵（后面用到）。

$$\mathring{q}\rightarrow R(\mathring{q})=\begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{bmatrix}$$

多个旋转的组合可以用四元数的乘法来实现。

$$R(\mathring{q}_0)\cdot R(\mathring{q}_1)=R(\mathring{q}_0\cdot\mathring{q}_1)$$

“四元数表示”转“欧拉角表示”。用于显示。

$$\mathring{q} \rightarrow \{\psi, \theta, \varphi\}$$

$$\begin{cases} \psi = \operatorname{atan2}(2wz + 2xy, 1 - 2y^2 - 2z^2) \\ \theta = \arcsin(2wy - 2zx) \\ \varphi = \operatorname{atan2}(2wx + 2yz, 1 - 2x^2 - 2y^2) \end{cases}$$

$$\begin{cases} \psi & \text{: 偏航角(yaw),z轴} \\ \theta & \text{: 俯仰角(pitch),y轴} \\ \varphi & \text{: 滚转角(roll),x轴} \end{cases}$$

姿态的测量

> 作者：nieyong

陀螺仪

陀螺仪，测量角速度，具有高动态特性，但是它是一个间接测量角度的器件，它测量的是角度的导数，角速度，要将角速度对时间积分才能得到角度。

如果这个世界是理想的，美好的，那我们的问题到此就解决了，不过很遗憾，现实是残酷的，误差的引入，使得积分出现了问题。

假设陀螺仪固定不动，理想角速度值是0dps(degree per second)，但是有一个偏置0.1dps加在上面，于是测量出来是0.1dps，积分一秒之后，得到的角度是0.1度，1分钟之后是6度，还能忍受，一小时之后是360度，转了一圈，也就是说，陀螺仪在短时间内有很大的参考价值。

使用陀螺仪获得角度，一定要考虑积分误差的问题。

加速度计

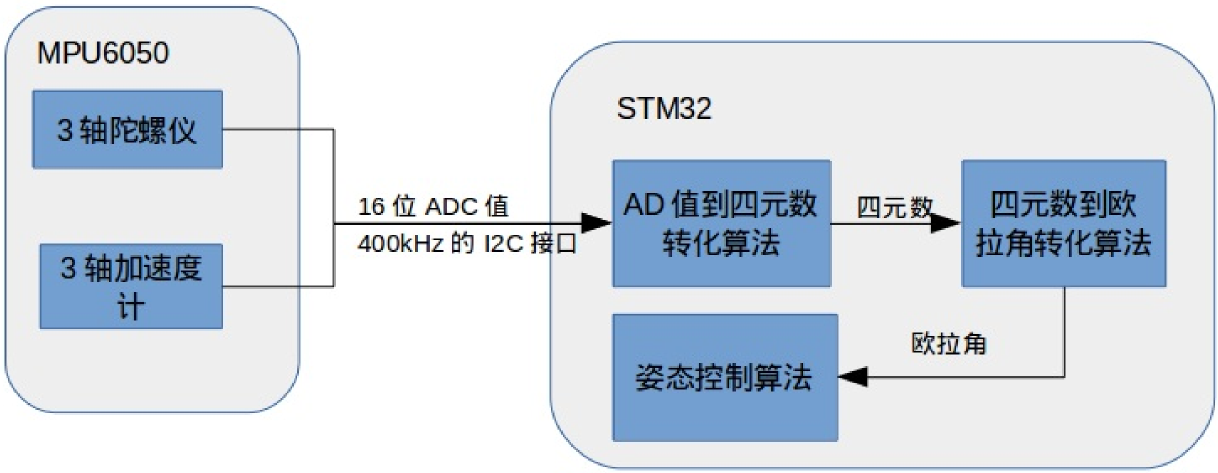
加速度计可以测量加速度,包括重力加速度,于是在静止或匀速运动的时候,加速度计仅仅测量的是重力加速度,而重力加速度与刚才所说的R坐标系是固连的,通过这种关系,可以得到加速度计所在平面 与 地面 的角度关系.

但是加速度计若是绕着重力加速度的轴转动,则测量值不会改变,也就是说无法感知这种水平旋转。

MPU6050姿态测量的两种方法

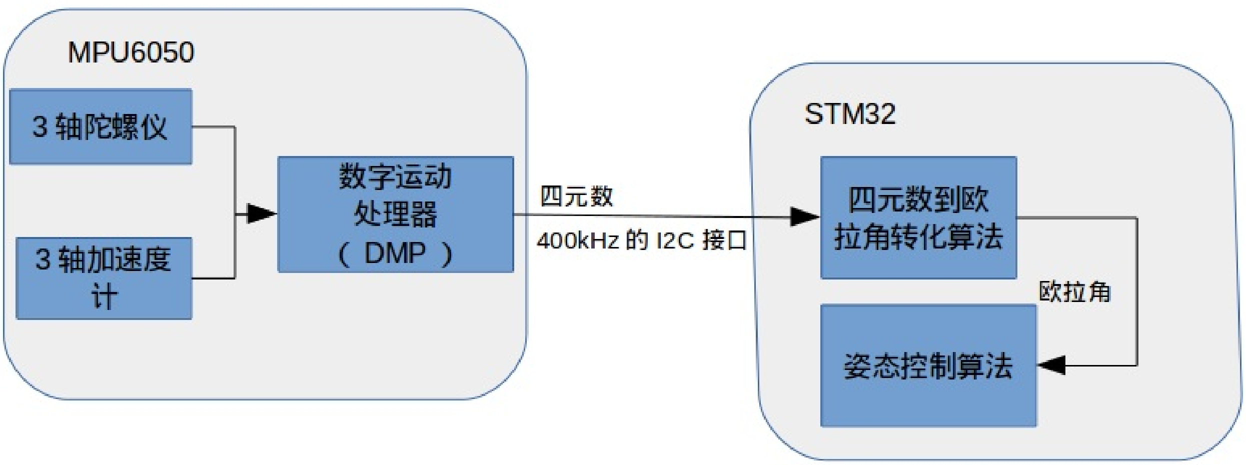
MPU-60x0是9轴运动处理传感器。它集成了3轴MEMS陀螺仪，3轴MEMS加速度计，以及一个可扩展的数字运动处理器DMP（Digital Motion Processor），可用I2C接口连接一个第三方的数字传感器，比如磁力计。扩展之后就可以通过其I2C或SPI接口输出一个9轴的信号。

MPU-60x0对陀螺仪和加速度计分别用了三个16位的ADC，将其测量的模拟量转化为可输出的数字量。在Crazepony-II上，测试了软件解算四元数，然后通过四元数解算姿态角这种实现方式，其实总的来说，并没感觉36MHz的主控压力有多大，没有出现机身不稳，卡死的情况。



同时，本着务实的态度，我们也测试了MPU6050的硬解四元数，即从IIC总线上读到的数据不再是MPU60x0的AD值，而是通过初始化对DMP引擎的配置，从IIC总线上读到的直接就是四元数的值，从而跳过了程序通过AD值计算四元数这个看起来繁琐的步骤。测试结果是，机身反应的确要比之前反应灵活，最关键的一点是，这样得出的偏航角（Yaw）很稳很稳，基本不会漂移或者说漂移小到了可以容忍的地步。

最后，MPU60x0的强大之处不仅于此，它支持一个从IIC接口，可以外部接上一个磁力计，如HMC5883，这样一来，DMP引擎可以直接输出一个绝对的方向姿态，即能够输出一个带东西南北的姿态数据包，很厉害的样子。在Crazepony-II第四版将会加上这样一个磁力计，相信它再也不会迷路了~~



姿态解算流程

> 作者：nieyong

姿态解算流程图

快速融合

长期融合

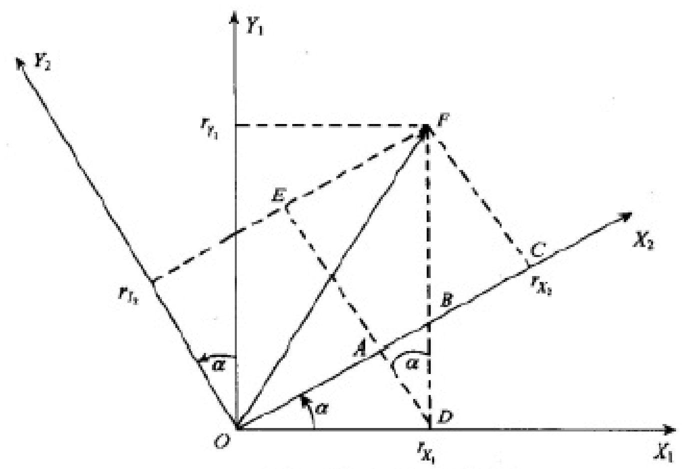
秦永元《惯性导航》 袁信、郑锷的《捷联式惯性导航原理》， 邓正隆的《惯性技术》

软件姿态解算

> 作者：路洋
>
> 文中有很多word下编辑的公式尚未加入，需要继续完善

使用MPU6050硬件DMP解算姿态是非常简单的，下面介绍由三轴陀螺仪和加速度计的值来使用四元数软件解算姿态的方法。

我们先来看看如何用欧拉角描述一次平面旋转(坐标变换)：



坐标系间的变换关系

设坐标系绕旋转α角后得到坐标系,在空间中有一个矢量在坐标系中的投影为,在内的投影为由于旋转绕进行，所以Z坐标未变，即有。

$$
\begin{aligned}
r_{x2} &= OA + AB + BC \\
&= OD\cos\alpha + BD\sin\alpha + BF\sin\alpha \\
&= r_{x1}\cos\alpha + r_{y1}\sin\alpha \\
r_{y2} &= DE - AD \\
&= DF\cos\alpha - OD\sin\alpha \\
&= r_{y1}\cos\alpha - OD\sin\alpha \\
&= r_{y1}\cos\alpha - r_{x1}\sin\alpha \\
r_{z2} &= r_{z1}
\end{aligned}
$$

转换成矩阵形式表示为：

$$\begin{bmatrix} r_{x2} \\ r_{y1} \\ r_{z1} \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{x1} \\ r_{y1} \\ r_{z1} \end{bmatrix}$$

整理一下：

$$r^1 = \begin{bmatrix} r_{x1} \\ r_{y1} \\ r_{z1} \end{bmatrix} \qquad r^2 = \begin{bmatrix} r_{x2} \\ r_{y2} \\ r_{z2} \end{bmatrix} \qquad \text{旋转阵 } C_1^2 = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

所以从旋转到可以写成

上面仅仅是绕一根轴的旋转，如果三维空间中的欧拉角旋转要转三次:

$$O-X_nY_nZ_n \xrightarrow{\text{绕}-Z_n\text{轴旋转}\,\psi} O-X_1Y_1Z_1 \xrightarrow{\text{绕}X_1\text{轴旋转}\,\theta} O-X_2Y_2Z_2 \xrightarrow{\text{绕}Y_2\text{轴旋转}\,\gamma} O-X_bY_bZ_b$$

$$C_n^b = C_2^b C_1^2 C_n^1 = \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & -\sin\phi & 0 \\ \sin\phi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\gamma\cos\psi + \sin\gamma\sin\psi\sin\theta & -\cos\gamma\sin\psi + \sin\gamma\cos\psi\sin\theta & -\sin\gamma\cos\theta \\ \sin\psi\cos\theta & \cos\psi\cos\theta & \sin\theta \\ \sin\gamma\cos\psi - \cos\gamma\sin\psi\sin\theta & -\sin\gamma\sin\psi - \cos\gamma\cos\psi\sin\theta & \cos\gamma\cos\theta \end{bmatrix}$$

上面得到了一个表示旋转的方向余弦矩阵。

不过要想用欧拉角解算姿态，其实我们套用欧拉角微分方程就行了：

$$\begin{bmatrix} \dot{\gamma} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \frac{1}{\cos\theta} \begin{bmatrix} \cos\theta & \sin\gamma\sin\theta & \cos\gamma\sin\theta \\ 0 & \cos\theta\cos\gamma & -\sin\gamma\cos\theta \\ 0 & \sin\gamma & \cos\gamma\cos\theta \end{bmatrix}^{-1} \bullet \begin{bmatrix} \omega_{EbX}^b \\ \omega_{EbY}^b \\ \omega_{EbZ}^b \end{bmatrix}$$

上式中左侧,,是本次更新后的欧拉角,对应row,pit,yaw。右侧，是上个周期测算出来的角度，，，三个角速度由直接安装在四轴飞行器的三轴陀螺仪在这个周期转动的角度，单位为弧度，计算间隔时T陀螺角速度，比如*0.02*秒0.01弧度/秒=0.0002弧度。间因此求解这个微分方程就能解算出当前的欧拉角。

前面介绍了什么是欧拉角，而且欧拉角微分方程解算姿态关系简单明了，概念直观容易理解，那么我们为什么不用欧拉角来表示旋转而要引入四元数呢？

一方面是因为欧拉角微分方程中包含了大量的三角运算，这给实时解算带来了一定的困难。而且当俯仰角为90度时方程式会出现神奇的“GimbalLock”。所以欧拉角方法只适用于水平姿态变化不大的情况，而不适用于全姿态飞行器的姿态确定。

四元数法只求解四个未知量的线性微分方程组，计算量小，易于操作，是比较实用的工程方法。

我们知道在平面(x,y)中的旋转可以用复数来表示，同样的三维中的旋转可以用单位四元数来描述。我们来定义一个四元数：

$$\mathbf{q} = a + \overrightarrow{u} = q_0 + q_1 i + q_2 j + q_3 k$$

我们可以把它写成,其中,。那么是矢量，表示三维空间中的旋转轴。w是标量，表示旋转角度。那么就是绕轴旋转w度，所以一个四元数可以表示一个完整的旋转。只有单位四元数才可以表示旋转，至于为什么，因为这就是四元数表示旋转的约束条件。

而刚才用欧拉角描述的方向余弦矩阵用四元数描述则为：

$$C_n^b = \begin{bmatrix} q_1^2 + q_0^2 - q_3^2 - q_2^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_2^2 - q_3^2 + q_0^2 - q_1^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_3^2 - q_2^2 - q_1^2 + q_0^2 \end{bmatrix}$$

所以在软件解算中，我们要首先把加速度计采集到的值(三维向量)转化为单位向量,即向量除以模，传入参数是陀螺仪x,y,z值和加速度计x,y,z值：

```
void IMUupdate(float gx, float gy, float gz, float ax, float ay, float az) {
float norm;
float vx, vy, vz;
float ex, ey, ez;

norm = sqrt(ax*ax + ay*ay + az*az);
ax = ax / norm;
ay = ay / norm;
az = az / norm;
```

下面把四元数换算成方向余弦中的第三行的三个元素。刚好vx,vy,vz 其实就是上一次的欧拉角（四元数）的机体坐标参考系换算出来的重力的单位向量。

```
// estimated direction of gravity
vx = 2*(q1*q3 - q0*q2);
vy = 2*(q0*q1 + q2*q3);
vz = q0*q0 - q1*q1 - q2*q2 + q3*q3;
```

axyz是机体坐标参照系上，加速度计测出来的重力向量，也就是实际测出来的重力向量。

axyz是测量得到的重力向量，vxyz是陀螺积分后的姿态来推算出的重力向量，它们都是机体坐标参照系上的重力向量。

那它们之间的误差向量，就是陀螺积分后的姿态和加计测出来的姿态之间的误差。

向量间的误差，可以用向量叉积（也叫向量外积、叉乘）来表示，exyz就是两个重力向量的叉积。

这个叉积向量仍旧是位于机体坐标系上的，而陀螺积分误差也是在机体坐标系，而且叉积的大小与陀螺积分误差成正比，正好拿来纠正陀螺。（你可以自己拿东西想象一下）由于陀螺是对机体直接积分，所以对陀螺的纠正量会直接体现在对机体坐标系的纠正。

```
// integral error scaled integral gain
exInt = exInt + ex*Ki;
eyInt = eyInt + ey*Ki;
ezInt = ezInt + ez*Ki;
```

用叉积误差来做PI修正陀螺零偏

```
// integral error scaled integral gain
exInt = exInt + ex*Ki;
eyInt = eyInt + ey*Ki;
ezInt = ezInt + ez*Ki;

// adjusted gyroscope measurements
gx = gx + Kp*ex + exInt;
gy = gy + Kp*ey + eyInt;
gz = gz + Kp*ez + ezInt;
```

四元数微分方程，其中T为测量周期，为陀螺仪角速度，以下都是已知量，这里使用了一阶龙哥库塔求解四元数微分方程：

$$O-X_nY_nZ_n \xrightarrow{\text{绕}-Z_n\text{轴旋转}\psi} O-X_1Y_1Z_1 \xrightarrow{\text{绕}X_1\text{轴旋转}\theta} O-X_2Y_2Z_2 \xrightarrow{\text{绕}Y_2\text{轴旋转}\gamma} O-X_bY_bZ_b$$

$$C_n^b = C_2^bC_1^2C_n^1 = \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} \cos\psi & -\sin\phi & 0 \\ \sin\phi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\gamma\cos\psi + \sin\gamma\sin\psi\sin\theta & -\cos\gamma\sin\psi + \sin\gamma\cos\psi\sin\theta & -\sin\gamma\cos\theta \\ \sin\psi\cos\theta & \cos\psi\cos\theta & \sin\theta \\ \sin\gamma\cos\psi - \cos\gamma\sin\psi\sin\theta & -\sin\gamma\sin\psi - \cos\gamma\cos\psi\sin\theta & \cos\gamma\cos\theta \end{bmatrix}$$

```
// integrate quaternion rate and normalise
q0 = q0 + (-q1*gx - q2*gy - q3*gz)*halfT;
q1 = q1 + (q0*gx + q2*gz - q3*gy)*halfT;
q2 = q2 + (q0*gy - q1*gz + q3*gx)*halfT;
q3 = q3 + (q0*gz + q1*gy - q2*gx)*halfT;
```

最后根据四元数方向余弦阵和欧拉角的转换关系，把四元数转换成欧拉角：

$$O-X_nY_nZ_n \xrightarrow{\text{绕}-Z_n\text{轴旋转}\psi} O-X_1Y_1Z_1 \xrightarrow{\text{绕}X_1\text{轴旋转}\theta} O-X_2Y_2Z_2 \xrightarrow{\text{绕}Y_2\text{轴旋转}\gamma} O-X_bY_bZ_b$$

$$C_n^b = C_2^bC_1^2C_n^1 = \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} \cos\psi & -\sin\phi & 0 \\ \sin\phi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\gamma\cos\psi + \sin\gamma\sin\psi\sin\theta & -\cos\gamma\sin\psi + \sin\gamma\cos\psi\sin\theta & -\sin\gamma\cos\theta \\ \sin\psi\cos\theta & \cos\psi\cos\theta & \sin\theta \\ \sin\gamma\cos\psi - \cos\gamma\sin\psi\sin\theta & -\sin\gamma\sin\psi - \cos\gamma\cos\psi\sin\theta & \cos\gamma\cos\theta \end{bmatrix}$$

所以有：

```
Q_ANGLE.Yaw = atan2(2 * q1 * q2 + 2 * q0 * q3, -2 * q2*q2 - 2 * q3* q3 + 1)* 57.3; // yaw
Q_ANGLE.Y  = asin(-2 * q1 * q3 + 2 * q0* q2)* 57.3; // pitch
Q_ANGLE.X = atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2* q2 + 1)* 57.3; // rol
```

硬件姿态解算

> 作者：路洋

四轴的姿态解算无疑是最繁琐的步骤没有之一，但是自从MPU6050出现了硬件DMP的时候，大妈都能完成姿态解算了！

CrazePony使用了MPU6050自带的硬件四元数单元，可以通过IIC直接读取四元数，省却了软件解算繁琐的算法步骤，非常方便易用。

这里还是要首先介绍下四元数，四元数要说的实在太多，因为它的优点很多，利用起来很方便，但是理解起来就有点蹩脚了。我们百度四元数，一开始看到的就是四元数来历，还有就是四元数的基本计算。对于来历，还是想说一下，四元数（Quaternions）是由威廉·卢云·哈密尔顿(William RowanHamilton,1805-1865）在1843 年爱尔兰发现的数学概念。

将实数域扩充到复数域，并用复数来表示平面向量，用复数的加、乘运算表示平面向量的合成、伸缩和旋，这就是我们熟知的复数的二维空间含义，所以人们会继续猜想，利用三维复数不就可以表达三维空间的变换了吗，历史上有很多数学家试图寻找过三维的复数，但后来证明这样的三维复数是不存在的。有关这个结论的证明，我没有查到更明确的版本，据《古今数学思想》中的一个理由，三维空间中的伸缩旋转变换需要四个变量来决定：两个变量决定轴的方向，一个变量决定旋转角度，一个变量决定伸缩比例。这样，只有三个变量的三维复数无法满足这样的要求。但是历史上得到的应该是比这个更强的结论，即使不考虑空间旋转，只从代数角度来说，三维的复数域作为普通复数域的扩张域是不存在的。并且，据《古今数学思想》叙述，即使像哈密尔顿后来引入四元数那样，牺牲乘法交换律，这样的三维复数也得不到。经过一些年的努力之后， Hamilton 发现自己被迫应作两个让步，第一个是他的新数包含四个分量，而第二个是他必须牺牲乘法交换律。（《古今数学思想》第三册177页）但是四元数用作旋转的作用明显，简化了运算，而且避免了Gimbal Lock，四元数是最简单的超复数，我们不能把四元数简单的理解为3D 空间的矢量，它是4 维空间中的的矢量，也是非常不容易想像的。

我们知道在平面(x,y)中的旋转可以用复数来表示，同样的三维中的旋转可以用单位四元数来描述。我们来定义一个四元数：

$$\mathbf{q} = a + \overrightarrow{u} = q_0 + q_1 i + q_2 j + q_3 k$$

我们可以把它写成$[w, \mathbf{v}]$，其中$\mathbf{v} = q_1 i + q_2 j + q_3 k$，$w = a = q_0$。那么V是矢量，表示三维空间中的旋转轴。w是标量，表示旋转角度。那么$[w, \mathbf{v}]$就是绕轴V旋转w度，所以一个四元数可以表示一个完整的旋转。只有单位四元数才可以表示旋转，至于为什么，因为这就是四元数表示旋转的约束条件。

所以大家可以理解为，单位四元数能够表示旋转。我们给出一组单位四元数和欧拉角的转换关系，通过这个关系来将采集到的四元数转化成欧拉角，原理将在软件解算中给出，这里直接使用就可以了：

$$\theta = -\sin^{-1} 2*(q_1 q_2 - q_0 q_3)$$

$$\psi = tg^{-1}\left(\frac{2(q_1 q_2 + q_0 q_3)}{q_1^2 + q_0^2 - q_3^2 - q_2^2}\right)$$

$$\gamma = tg^{-1}\left(\frac{2(q_1 q_3 + q_0 q_1)}{q_3^2 + q_2^2 - q_1^2 + q_0^2}\right)$$

所以在四轴飞行器中，传感器读取到四元数，首先应先将它归一化成单位四元数：

```
norm = dmpinvSqrt(q[0]*q[0] + q[1]*q[1] + q[2]*q[2] + q[3]*q[3]);
q[0] = q[0] * norm;
q[1] = q[1] * norm;
q[2] = q[2] * norm;
q[3] = q[3] * norm;
```

归一化后根据四元数和欧拉角转换公式把四元数转化为欧拉角，OK,硬件姿态解算完成！

```
DMP_DATA.dmp_roll = (atan2(2.0*(q[0]*q[1] + q[2]*q[3]), 1 - 2.0*(q[1]*q[1] + q[2]*q[2])))
// we let safe_asin() handle the singularities near 90/-90 in pitch
DMP_DATA.dmp_pitch = dmpsafe_asin(2.0*(q[0]*q[2] - q[3]*q[1]))* 180/M_PI;

DMP_DATA.dmp_yaw = -atan2(2.0*(q[0]*q[3] + q[1]*q[2]), 1 - 2.0*(q[2]*q[2] + q[3]*q[3]))*
```

四元数与四轴飞行器

> 作者：nieyong

四元数

四元数是由爱尔兰数学家威廉·卢云·哈密顿在1843年发现的数学概念。从明确地角度而言，四元数是复数的不可交换延伸。如把四元数的集合考虑成多维实数空间的话，四元数就代表着一个四维空间，相对于复数为二维空间。

四元数大量用于电脑绘图（及相关的图像分析）上表示三维物件的旋转及方位。四元数亦见于控制论、信号处理、姿态控制、物理和轨道力学，都是用来表示旋转和方位。

相对于另几种旋转表示法（矩阵，欧拉角，轴角），四元数具有某些方面的优势，如速度更快、提供平滑插值、有效避免万向锁问题、存储空间较小等等。

以上部分摘自维基百科-四元数。

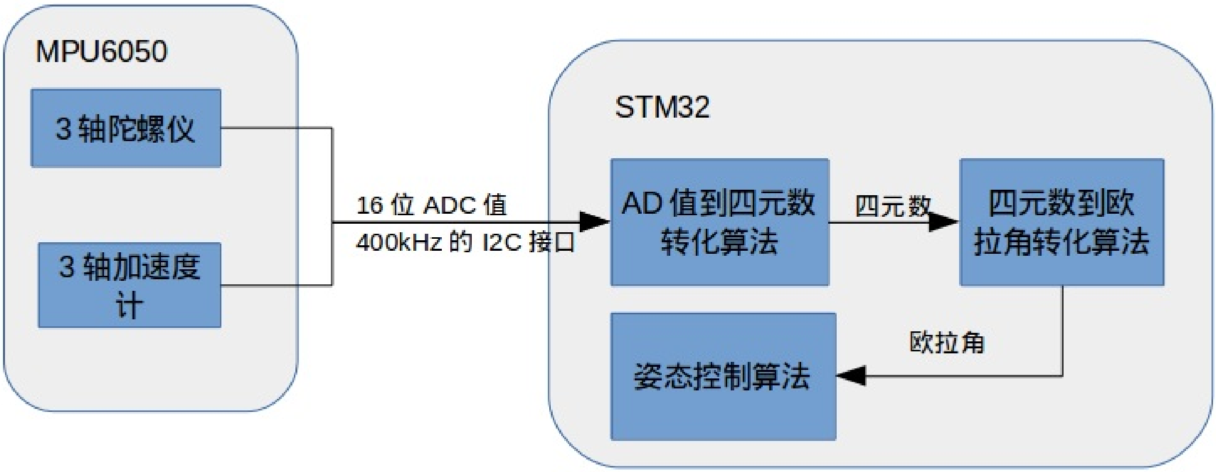
姿态解算为什么要用四元数

姿态解算的核心在于旋转，一般旋转有4种表示方式：矩阵表示、欧拉角表示、轴角表示和四元数表示。矩阵表示适合变换向量，欧拉角最直观，轴角表示则适合几何推导，而在组合旋转方面，四元数表示最佳。因为姿态解算需要频繁组合旋转和用旋转变换向量，所以采用四元数保存组合姿态、辅以矩阵来变换向量的方案。

总结来说，在crazepony中，姿态解算中使用四元数来保存飞行器的姿态，包括旋转和方位。在获得四元数之后，会将其转化为欧拉角，然后输入到姿态控制算法中。

姿态控制算法的输入参数必须要是欧拉角。AD值是指MPU6050的陀螺仪和加速度值，3个维度的陀螺仪值和3个维度的加速度值，每个值为16位精度。AD值必须先转化为四元数，然后通过四元数转化为欧拉角。这个四元数可能是软解，主控芯片（STM32）读取到AD值，用软件从AD值算得，也可能是通过MPU6050中的DMP硬解，主控芯片（STM32）直接读取到四元数。具体参考《MPU60x0的四元数生成方式介绍》。

下面就是四元数软解过程，可以由下面这个框图表示：



四元数的运算

下面介绍一下四元数，然后给出几种旋转表示的转换。这些运算在crazepony的代码中都会遇到。

四元数可以理解为一个实数和一个向量的组合，也可以理解为四维的向量。这里用一个圈表示q是一个四元数（很可能不是规范的表示方式）。

$$\mathring{q}=\{w,\vec{v}\}=[w \quad x \quad y \quad z]^T$$

四元数的长度（模）与普通向量相似。

$$|\mathring{q}|=\sqrt{w^2+x^2+y^2+z^2}$$

下面是对四元数的单位化，单位化的四元数可以表示一个旋转。

$$\hat{\mathring{q}}=\frac{\mathring{q}}{|\mathring{q}|}=\left[\frac{w}{|\mathring{q}|} \quad \frac{x}{|\mathring{q}|} \quad \frac{y}{|\mathring{q}|} \quad \frac{z}{|\mathring{q}|}\right]^T$$

四元数相乘，旋转的组合就靠它了。

$$\mathring{q}_0=\mathring{q}_1\cdot\mathring{q}_2$$

$$\begin{cases} w_0=w_1w_2-x_1x_2-y_1y_2-z_1z_2 \\ x_0=w_1x_2+x_1w_2+y_1z_2-z_1y_2 \\ y_0=w_1y_2-x_1z_2+y_1w_2+z_1x_2 \\ z_0=w_1z_2+x_1y_2-y_1x_2+z_1w_2 \end{cases}$$

旋转的“轴角表示”转“四元数表示”。这里创造一个运算q(w,θ)，用于把绕单位向量w转θ角的旋转表示为四元数。

$$\{\hat{w},\theta\}\rightarrow\mathring{q}(\hat{w},\theta)=\{\cos(\frac{\theta}{2}),\hat{w}\cdot\sin(\frac{\theta}{2})\}$$

通过q(w,θ)，引伸出一个更方便的运算q(f,t)。有时需要把向量f的方向转到向量t的方向，这个运算就是生成表示对应旋转的四元数的（后面会用到）。

$$\mathring{q}(\vec{f},\vec{t})=\mathring{q}(\frac{\vec{f}\times\vec{t}}{|\vec{f}\times\vec{t}|},\mathrm{atan2}(|\vec{f}\times\vec{t}|,\vec{f}\cdot\vec{t}))$$

然后是“四元数表示”转“矩阵表示”。再次创造运算，用R(q)表示四元数q对应的矩阵（后面用到）。

$$\mathring{q}\rightarrow R(\mathring{q})=\begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{bmatrix}$$

多个旋转的组合可以用四元数的乘法来实现。

$$R(\mathring{q}_0)\cdot R(\mathring{q}_1)=R(\mathring{q}_0\cdot\mathring{q}_1)$$

“四元数表示”转“欧拉角表示”。用于显示。

$$\mathring{q} \rightarrow \{\psi, \theta, \varphi\}$$

$$\begin{cases} \psi = \operatorname{atan2}(2wz + 2xy, 1 - 2y^2 - 2z^2) \\ \theta = \arcsin(2wy - 2zx) \\ \varphi = \operatorname{atan2}(2wx + 2yz, 1 - 2x^2 - 2y^2) \end{cases}$$

$$\begin{cases} \psi & : \text{偏航角(yaw),z轴} \\ \theta & : \text{俯仰角(pitch),y轴} \\ \varphi & : \text{滚转角(roll),x轴} \end{cases}$$

三轴陀螺仪三轴加速度计MPU6050

> 作者：nieyong

陀螺仪

陀螺仪，测量角速度，具有高动态特性，但是它是一个间接测量角度的器件，它测量的是角度的导数，角速度，要将角速度对时间积分才能得到角度。

如果这个世界是理想的，美好的，那我们的问题到此就解决了，不过很遗憾，现实是残酷的，误差的引入，使得积分出现了问题。

假设陀螺仪固定不动，理想角速度值是0dps(degree per second)，但是有一个偏置0.1dps加在上面，于是测量出来是0.1dps，积分一秒之后，得到的角度是0.1度，1分钟之后是6度，还能忍受，一小时之后是360度，转了一圈，也就是说，陀螺仪在短时间内有很大的参考价值。

陀螺仪就是内部有一个陀螺，它的轴由于陀螺效应始终与初始方向平行，这样就可以通过与初始方向的偏差计算出实际方向。传感器MPU6050实际上是一个结构非常精密的芯片，内部包含超微小的陀螺。陀螺仪运转一段时间以后，noise和offset会导致数据偏差，需要借助其它传感器进行较正。

使用陀螺仪获得角度，一定要考虑积分误差的问题。

加速度计

加速度计可以测量加速度,包括重力加速度,于是在静止或匀速运动的时候,加速度计仅仅测量的是重力加速度,而重力加速度与刚才所说的R坐标系是固连的,通过这种关系,可以得到加速度计所在平面 与 地面 的角度关系.

但是加速度计若是绕着重力加速度的轴转动,则测量值不会改变,也就是说无法感知这种水平旋转。

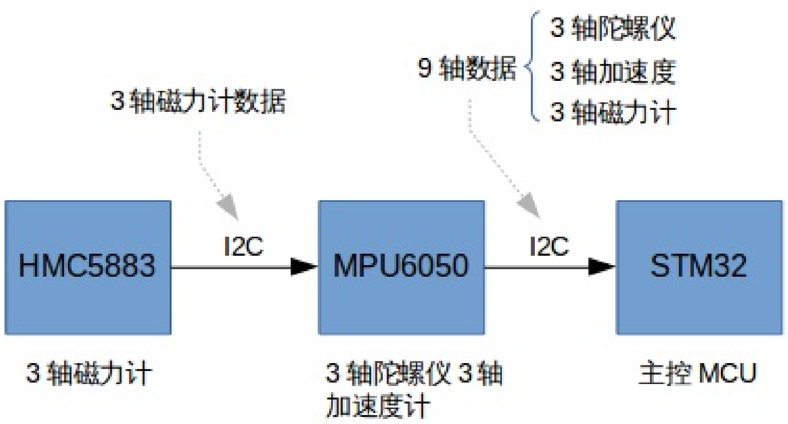
关于MPU6050

MPU-60x0是全球首例9轴运动处理传感器。它集成了3轴MEMS陀螺仪，3轴MEMS加速度计，以及一个可扩展的数字运动处理器DMP（Digital Motion Processor），可用I2C接口连接一个第三方的数字传感器，比如磁力计。扩展之后就可以通过其I2C或SPI接口输出一个9轴的

信号（SPI接口仅在MPU-6000可用）。MPU-60x0也可以通过其I2C接口连接非惯性的数字传感器，比如压力传感器。

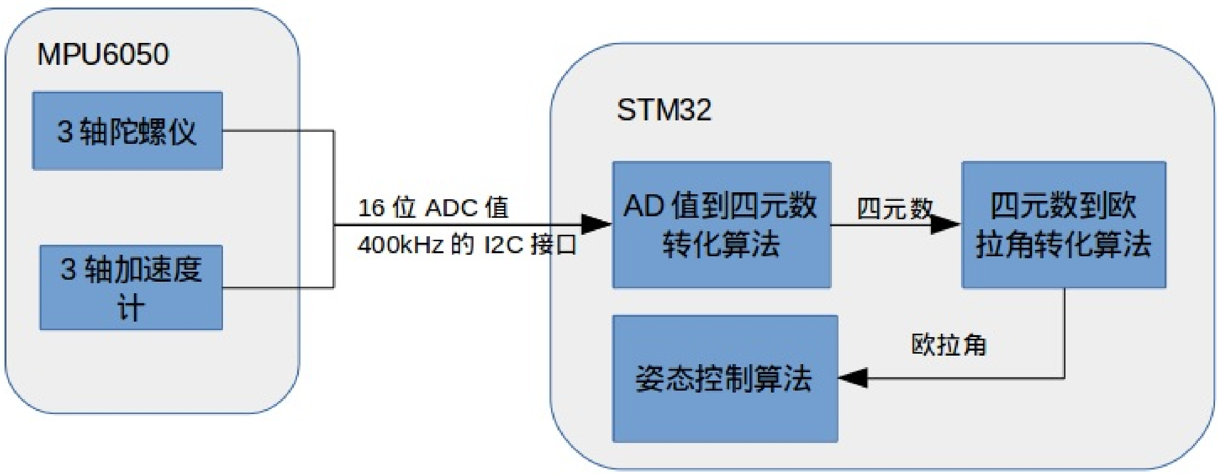
MPU-60x0对陀螺仪和加速度计分别用了三个16位的ADC，将其测量的模拟量转化为可输出的数字量。为了精确跟踪快速和慢速的运动，传感器的测量范围都是用户可控的，陀螺仪可测范围为±250，±500，±1000，±2000°/秒（dps），加速度计可测范围为±2，±4，±8，±16g。一个片上1024 字节的FIFO，有助于降低系统功耗。和所有设备寄存器之间的通信采用400kHz的I2C接口或1MHz 的SPI接口（SPI仅MPU-6000可用）。对于需要高速传输的应用，对寄存器的读取和中断可用20MHz的SPI。另外，片上还内嵌了一个温度传感器和在工作环境下仅有±1%变动的振荡器。

在crazepony上，MPU6050,HMC5883传感器之间的连接如下图所示。



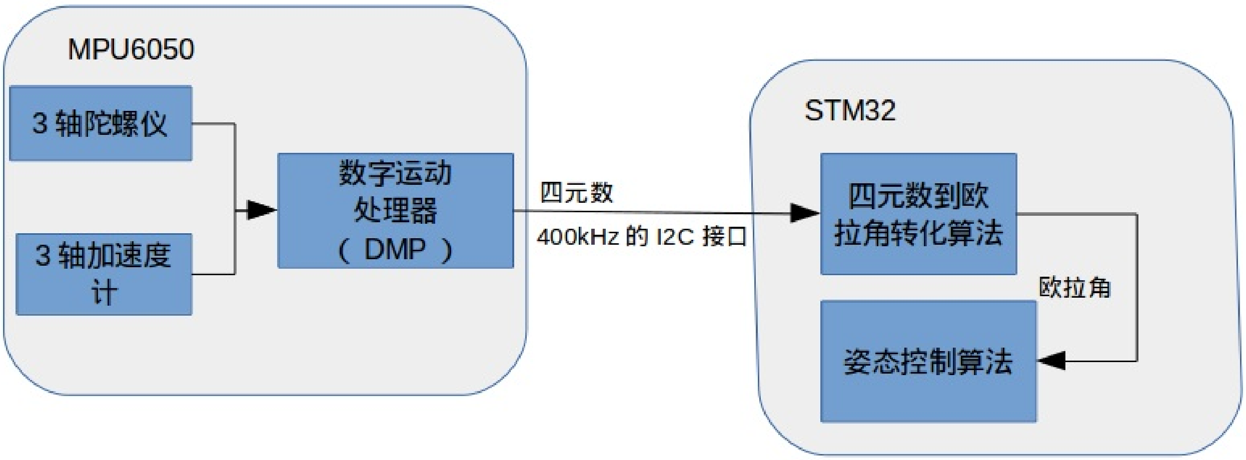
MPU6050中DMP应用

在Crazepony-II上，测试了软件解算四元素，然后通过四元素解算姿态角这种实现方式，其实总的来说，并没感觉36MHz的主控压力有多大，没有出现机身不稳，卡死的情况。



同时，本着务实他的态度，我们也测试了MPU6050的硬解四元素，即从IIC总线上读到的数据不再是MPU60x0的AD值，而是通过初始化对DMP引擎的配置，从IIC总线上读到的直接就是四元素的值，从而跳过了程序通过AD值计算四元素这个看起来繁琐的步骤。测试结果是，机身反应的确要比之前反应灵活，最关键的一点是，这样得出的偏航角（Yaw）很稳很稳，基本不会漂移或者说漂移小到了可以容忍的地步。

最后，MPU60x0的强大之处不仅于此，它支持一个从IIC接口，可以外部接上一个磁力计，如HMC5883，这样一来，DMP引擎可以直接输出一个绝对的方向姿态，即能够输出一个带东西南北的姿态数据包，很厉害的样子。在Crazepony-II第四版将会加上这样一个磁力计，相信它再也不会迷路了~~



四轴PID控制算法

> 作者：nieyong

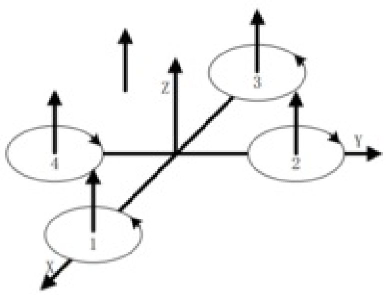
四轴控制原理

四轴飞行器的螺旋桨与空气发生相对运动，产生了向上的升力，当升力大于四轴的重力时四轴就可以起飞了。

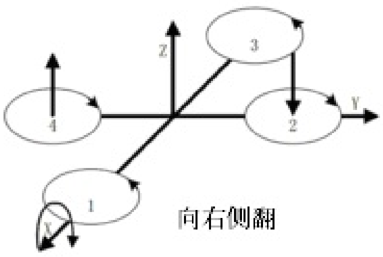
四轴飞行器飞行过程中如何保持水平呢？

我们先假设一种理想状况，四个电机的转速是完全相同的，是不是我们控制四轴飞行器的四个电机保持同样的转速，当转速超过一个临界点时（升力刚好抵消重力）四轴就可以平稳的飞起来了呢？

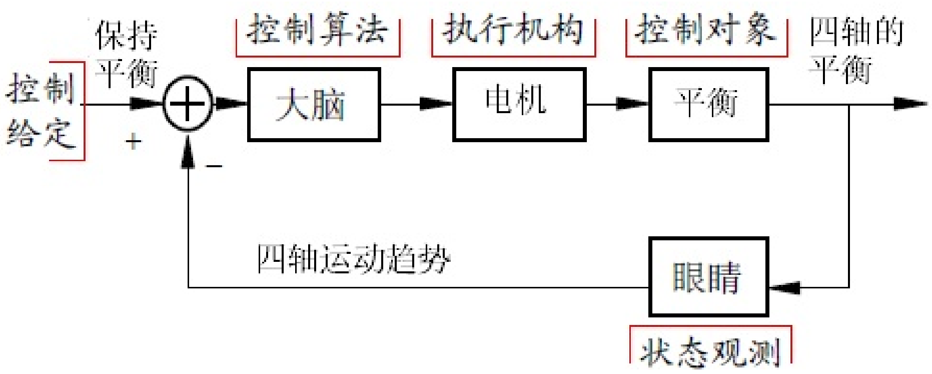
答案是否定的，由于四个电机转向相同，四轴会发生旋转。我们控制四轴电机1和电机3同向（逆时针旋转），电机2电机4反向（顺时针旋转），刚好正反扭矩抵消，巧妙的实现了平衡，如下图所示。



实际上由于电机和螺旋桨本身制造的差异我们无法做到四个电机转速完全相同，很有可能飞行器起飞之后就侧翻。这时候大家可能会想到要用遥控器来控制电机，我们来尝试一下下面向右侧翻的情况。



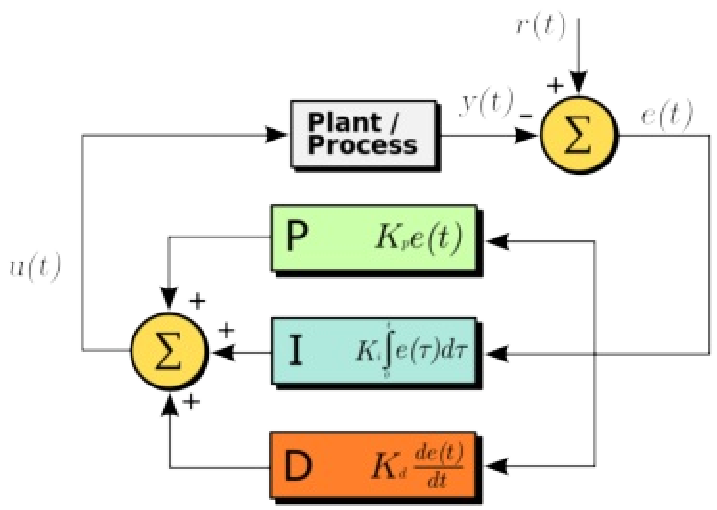
由于电机的不平衡，在人眼的观察下发现飞机向右侧翻，我们控制右侧电机1电机2提高转速增加升力，飞机归于平衡。由于 飞机是一个动态系统，在接下来我们会一直重复：观察->大脑计算->控制->观察->大脑计算->控制。



但事实上这是不可能的，因为人无法长时间精确的同时控制四个电机。我们需要一个自动反馈系统替代人操作来完成飞机的自稳定，我们人只需要控制飞机的方向和高度就可以了。这个系统中反馈由姿态传感器替代眼睛，而大脑则由单片机来替代。这时候该PID控制系统出场。

PID控制理论

PID控制是最常见，应用最为广泛的自动反馈系统。PID控制器由偏差的比例（P，Proportional）、积分（I，Integral）和微分（D，Derivative）来对被控对象进行控制。这里的积分或微分，都是偏差对时间的积分或微分。



对于一个自动反馈控制系统来说，有几个基本的指标。

- 稳定性（P和I降低系统稳定性，D提高系统稳定性）：在平衡状态下，系统受到某个干扰后，经过一段时间其被控量可以达到某一稳定状态；
- 准确性（P和I提高稳态精度，D无作用）：系统处于稳态时，其稳态误差（Steady-state error）；
- 快速性（P和D提高响应速度，I降低响应速度）：系统对动态响应的要求。一般由过渡时间的长短来衡量。

比例（P）控制

比例控制是一种最简单的控制方式。其控制器的输出与输入误差信号成比例关系。当仅有比例控制时系统输出存在稳态误差。比例项输出：

$$P_{\text{out}} = K_p\, e(t)$$

积分（I）控制

在积分控制中，控制器的输出与输入误差信号的积分成正比关系。对于只有比例控制的系统存在稳态误差，为了消除稳态误差，在控制器中必须引入“积分项”。积分项是误差对时间的积分，随着时间的增加，积分项会增大。这样，即便误差很小，积分项也会随着时间的增加而加大，它推动控制器的输出增大使稳态误差进一步减小，直到等于零。因此，比例积分(PI)控制器，可以使系统在进入稳态后无稳态误差。 积分项输出：

$$I_{\text{out}} = K_i \int_0^t e(\tau)\, d\tau$$

微分（D）控制

在微分控制中，控制器的输出与输入误差信号的微分成正比关系。微分调节就是偏差值的变化率。使用微分调节能够实现系统的超前控制。如果输入偏差值线性变化，则在调节器输出侧叠加一个恒定的调节量。大部分控制系统不需要调节微分时间。因为只有时间滞后的系统才需要附加这个参数。微分项输出：

$$D_{\text{out}} = K_d \frac{d}{dt} e(t)$$

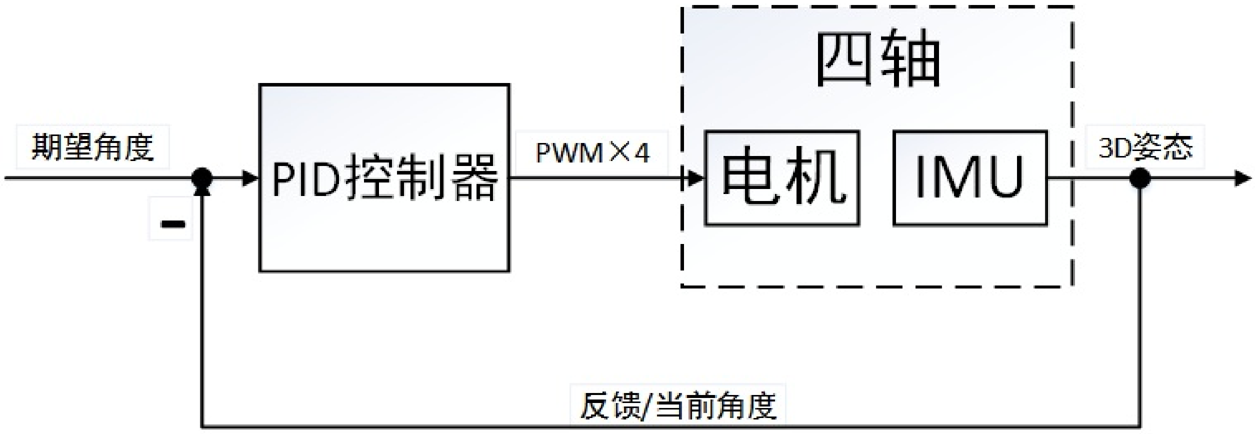
综上所述得到一条公式，这个就是PID控制数学表达式：

$$\mathrm{u}(t) = \mathrm{MV}(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{d}{dt} e(t)$$

更多关于PID控制理论知识，可以参考维基百科PID控制。

四轴PID控制-单环

在Crazepony四轴飞行器5.0版本及以前，我们使用的是单环增量式PD控制，下面是角度单环PID控制框图。这里对图中几个数据进行说明，期望角度就是遥控器控制飞行器的角度值，反馈当前角度就是传感器测得的飞行器角度，这里的角度指的是Roll/Pitch/Yaw三个角度，而且在PID控制计算的时候，是相互独立的。



角度单环PID原理框图 //blog.csdn.net/nemol1990

以ROLL方向角度控制为例：

- 测得ROLL轴向偏差：

```
偏差=目标期望角度-传感器实测角度
DIF_ANGLE.X = EXP_ANGLE.X - Q_ANGLE.Roll;
```

- 比例项的计算：

```
比例项输出 = 比例系数P * 偏差
Proportion =  PID_Motor.P * DIF_ANGLE.X;
```

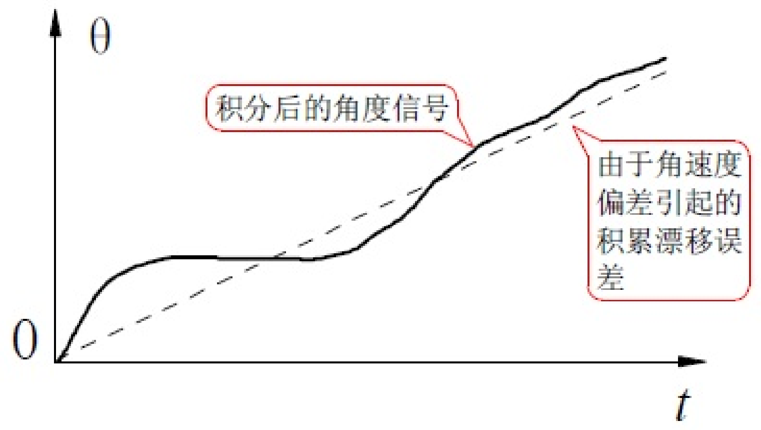
- 微分项计算： 由于陀螺仪测得的是ROLL轴向旋转角速率，角速率积分就是角度，那么角度微分即角速率，所以微分量刚好是陀螺仪测得的值。

```
微分输出=微分系数D*角速率
DifferentialCoefficient = PID_Motor.D * DMP_DATA.GYROx;
```

- 整合结果总输出为：

```
ROLL方向总控制量=比例项输出+微分量输出
```

ROLL和PITCH轴按照以上公式计算PID输出，但YAW轴比较特殊，因为偏航角法线方向刚好和地球重力平行，这个方向的角度无法由加速度计直接测得，需要增加一个电子罗盘来替代加速度计。如果不使用罗盘的话，我们可以单纯的通过角速度积分来测得偏航角，缺点是由于积分环节中存在积分漂移，偏航角随着时间的推移会偏差越来越大，就会出现航向角漂移的问题。我们不使用罗盘就没有比例项，只仅使用微分环节来控制。



角速度积分漂移现象

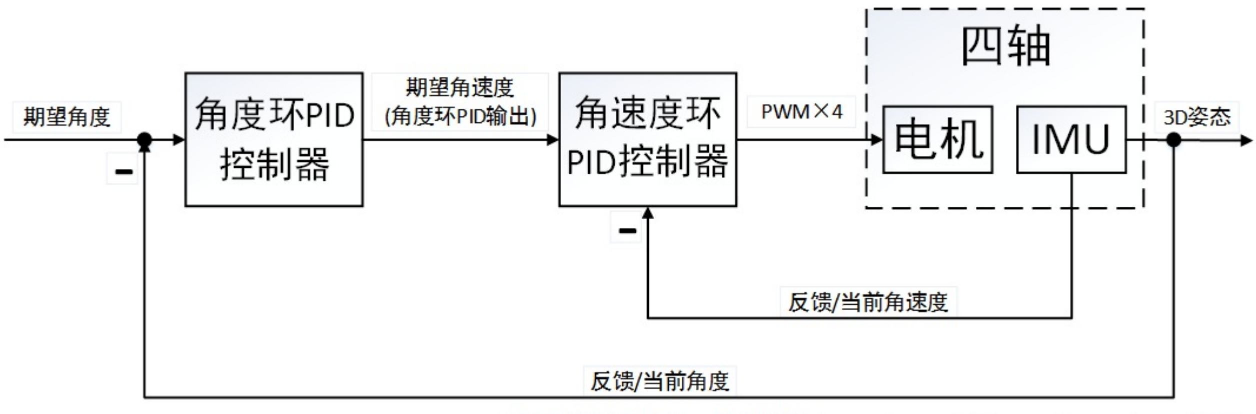
- YAW轴输出：

```
微分输出=微分系数D*角速率

YAW方向控制量 = PID_YAW.D * DMP_DATA.GYROz;
```

四轴PID控制-串级

角度单环PID控制算法仅仅考虑了飞行器的角度信息，如果想增加飞行器的稳定性(增加阻尼)并提高它的控制品质，我们可以进一步的控制它的角速度，于是角度/角速度-串级PID控制算法应运而生。在这里，相信大多数朋友已经初步了解了角度单环PID的原理，但是依旧无法理解串级PID究竟有什么不同。其实很简单：它就是两个PID控制算法，只不过把他们串起来了(更精确的说是套起来)。那这么做有什么用？答案是，它增强了系统的抗干扰性(也就是增强稳定性)，因为有两个控制器控制飞行器，它会比单个控制器控制更多的变量，使得飞行器的适应能力更强。为了更为清晰的讲解串级PID，这里笔者依旧画出串级PID的原理框图，如图所示：



角度/角速度-串级PID原理框图

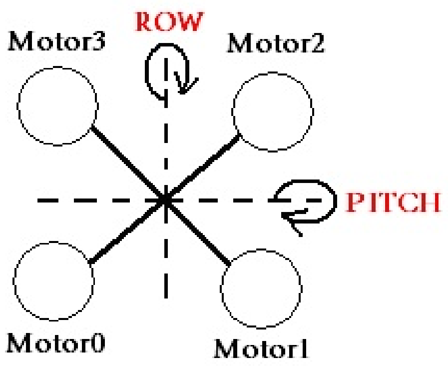
Crazepony在5.1版本及以后，就是采用角度/角速度串级PID控制。详见 `Control.c` 文件中的 `CtrlAttiAng(void)` 函数和 `CtrlAttiRate(void)` 函数。

如何整定串级PID时的经验，来自CSDN网友Nemo之家博客四轴PID讲解。原则是先整定内环PID，再整定外环P。

- 内环P：从小到大，拉动四轴越来越困难，越来越感觉到四轴在抵抗你的拉动；到比较大的数值时，四轴自己会高频震动，肉眼可见，此时拉扯它，它会快速的振荡几下，过几秒钟后稳定；继续增大，不用加人为干扰，自己发散翻机。特别注意：只有内环*P*的时候，四轴会缓慢的往一个方向下掉，这属于正常现象。这就是系统角速度静差。
- 内环I：前述PID原理可以看出，积分只是用来消除静差，因此积分项系数个人觉得没必要弄的很大，因为这样做会降低系统稳定性。从小到大，四轴会定在一个位置不动，不再往下掉；继续增加I的值，四轴会不稳定，拉扯一下会自己发散。特别注意：增加*I*的值，四轴的定角度能力很强，拉动他比较困难，似乎像是在钉钉子一样，但是一旦有强干扰，它就会发散。这是由于积分项太大，拉动一下积分速度快，给 的补偿非常大，因此很难拉动，给人一种很稳定的错觉。
- 内环D：这里的微分项D为标准的PID原理下的微分项，即本次误差-上次误差。在角速度环中的微分就是角加速度，原本四轴的震动就比较强烈，引起陀螺的值变化较大，此时做微分就更容易引入噪声。因此一般在这里可以适当做一些滑动滤波或者IIR滤波。从小到大，飞机的性能没有多大改变，只是回中的时候更加平稳；继续增加D的值，可以肉眼看到四轴在平衡位置高频震动(或者听到电机发出滋滋的声音)。前述已经说明D项属于辅助性项，因此如果机架的震动较大，D项可以忽略不加。
- 外环P：当内环PID全部整定完成后，飞机已经可以稳定在某一位置而不动了。此时内环P，从小到大，可以明显看到飞机从倾斜位置慢慢回中，用手拉扯它然后放手，它会慢速回中，达到平衡位置；继续增大P的值，用遥控器给不同的角度给定，可以看到飞机跟踪的速度和响应越来越快；继续增加P的值，飞机变得十分敏感，机动性能越来越强，有发散的趋势。

电机的输出整合

油门控制Throttle是电机输出的基准值，增加油门即可四轴升高，减小油门即下降。最后整合上面通过PID算得的ROLL/PITCH/YAW三轴输出量进行电机控制。整合如下，在 `Control.c` 文件的函数 `CtrlMotor()` 中实现。




```
Motor[2] = (int16_t)(Thr - Pitch - Roll - Yaw );    //M3
Motor[0] = (int16_t)(Thr + Pitch + Roll - Yaw );    //M1
Motor[3] = (int16_t)(Thr - Pitch + Roll + Yaw );    //M4
Motor[1] = (int16_t)(Thr + Pitch - Roll + Yaw );    //M2
```

Crazepony左下角为M1机臂，在代码中对应电机为 `Motor[0]` ，然后逆时针依次为M2，M3，M4，如图所示。如果四轴绕ROLL轴向右倾斜5度，那么电机1电机2应该提高升力，电机3电机0减小升力恢复平衡状态，所以有以下规则：

- Roll方向旋转，为了恢复平衡，则电机1电机2同侧出力，电机0电机3反向出力（Motor[1]和Motor[2]中的Roll为 `-` ，Motor[0]和Motor[3]中的Roll为 `+` ）
- Pitch方向旋转，为了恢复平衡，则电机2电机3同侧出力，电机0电机1反向出力（Motor[2]和Motor[3]中的Pitch为 `-` ，Motor[0]和Motor[1]中的Pitch为 `+` ）
- Yaw方向旋转，为了恢复平衡，则电机1电机3同侧出力，电机0电机2反向出力，（Motor[1]和Motor[3]中的Yaw为 `+` ，Motor[0]和Motor[2]中的Yaw为 `-` ）

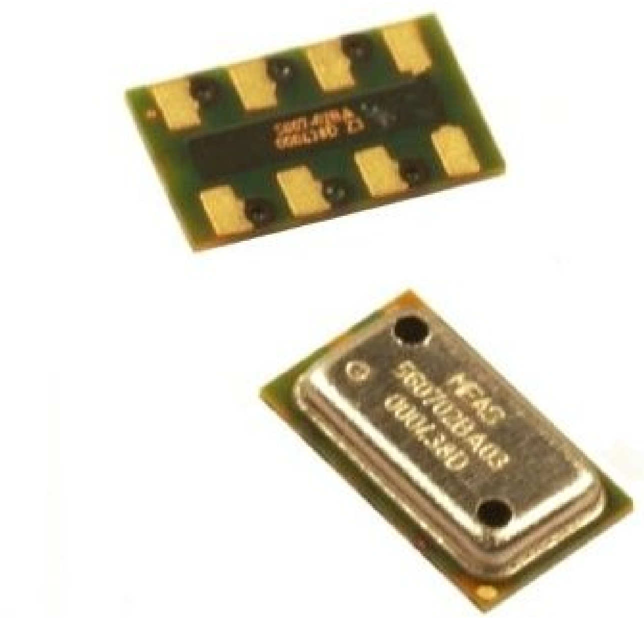
本篇内容，部分来自Nemo之家博客四轴PID讲解。

气压计MS5611

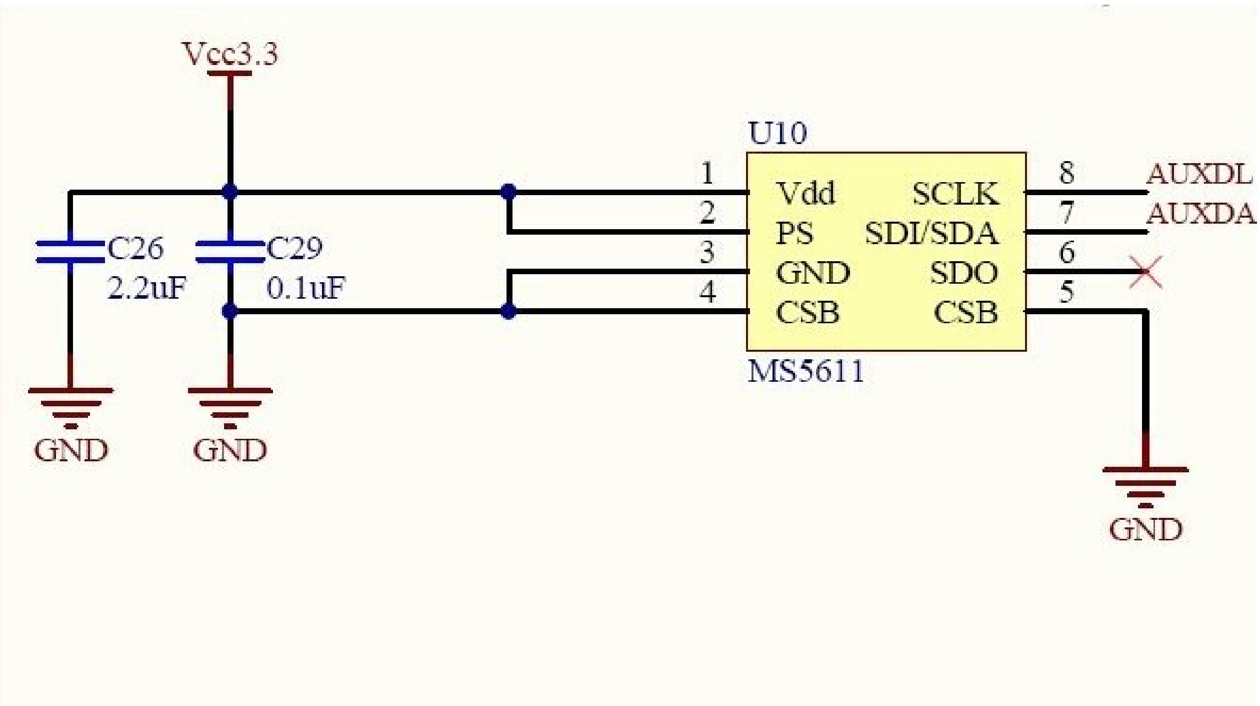
> 作者：nieyong

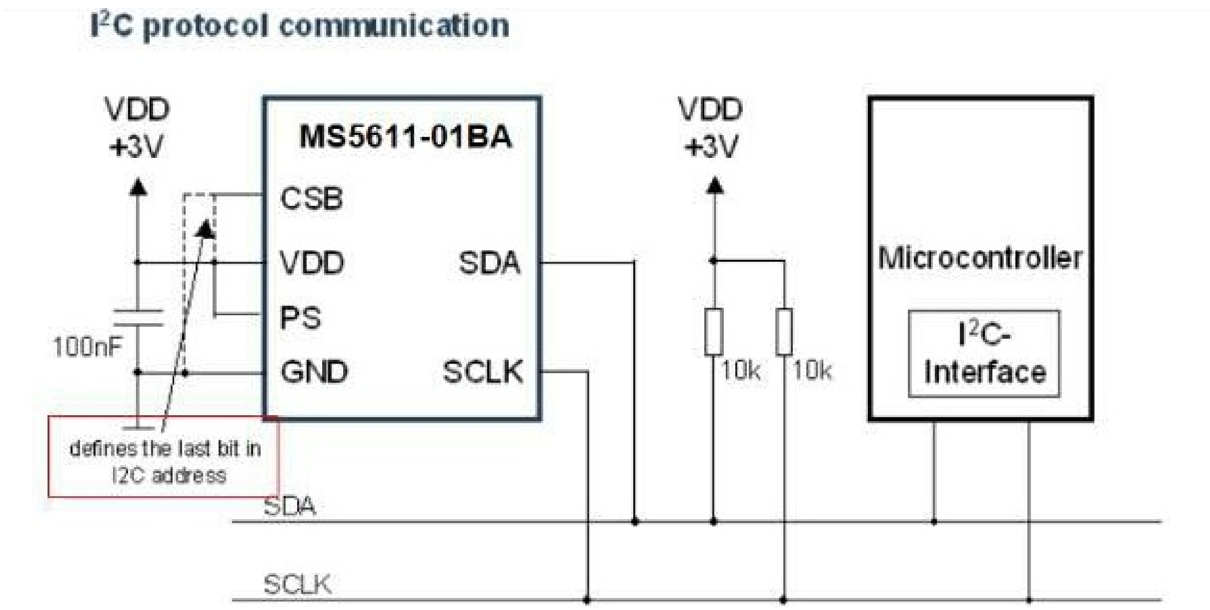
Crazepony使用气压计（或者叫高度计）MS5611。

MS5611气压传感器是由MEAS（瑞士）推出的SPI和I²C总线接口的新一代高分辨率气压传感器，分辨率可达到**10cm**。我们使用的I2C接口，挂在MPU5060的从I2C接口上。MS5611有2种型号的封装MS5611-01BA03(金属封装) 和MS5611-01BA01(塑壳封装），我们使用的金属封装。



MS5611的I2C地址为0b111011Cx，其中C比特位由CSB引脚决定，为CSB引脚的补码值（取反）。Crazepony上MS5611的CSB引脚接地，所以CSB引脚值为0，8位I2C地址为0b1110111x（0xEE），7位I2C地址为0b1110111（0x77）。





MS5611的I2C操作命令

下面以I2C连接模式讲解，符合Crazepony硬件上的连接。对MS5611芯片操作，只有5个命令。

- 复位芯片

复位MS5611芯片，写入一个字节0x1E即可。

```
IICwriteOneByte( ms5611Address, 0x1E);      // Reset Device
```

- 读取PROM中的值

MS5611气压传感器在出厂的时候，将该片IC对应的6个系数写在了内部寄存器（PROM）中，读取该系数，用于后面对实际温度和实际气压的计算。

内部寄存器的位置分别为0xA2/0xA4/0xA6/0xA8/0xAA/0xAC/0xAE，只需要对该位置读取2个字节则可。

```
IICreadBytes( ms5611Address, 0xA2, 2, data);    // Read Calibration Data C1
c1.bytes[1] = data[0];
c1.bytes[0] = data[1];

IICreadBytes( ms5611Address, 0xA4, 2, data);    // Read Calibration Data C2
c2.bytes[1] = data[0];
c2.bytes[0] = data[1];

IICreadBytes( ms5611Address, 0xA6, 2, data);    // Read Calibration Data C3
c3.bytes[1] = data[0];
c3.bytes[0] = data[1];

IICreadBytes( ms5611Address, 0xA8, 2, data);    // Read Calibration Data C4
c4.bytes[1] = data[0];
c4.bytes[0] = data[1];

IICreadBytes( ms5611Address, 0xAA, 2, data);    // Read Calibration Data C5
c5.bytes[1] = data[0];
c5.bytes[0] = data[1];

IICreadBytes( ms5611Address, 0xAC, 2, data);    // Read Calibration Data C6
c6.bytes[1] = data[0];
c6.bytes[0] = data[1];
```

- 配置进行D1转化

D1转化即气压转化，告知IC进行气压转化，下一步就可以读取气压值（24位）。如果OSR为4096，则写入0x48即可。

```
IICwriteOneByte( ms5611Address, 0x48);
```

- 配置进行D2转化

D2转化即温度转化，告知IC进行温度转化，下一步就可以读取温度值（24位）。如果OSR为4096，则写入0x58即可。

```
IICwriteOneByte( ms5611Address, 0x58);
```

- 读取ADC的值

ADC值即上面进行的D1转化或者D2转化，为24位数值。对0x00读取3个字节即可。

```
IICreadBytes( ms5611Address, 0x00, 3, data);    // Request temperature read
```

上面这5个命令，是对ms5611操作的所有命令了。我们可以获得6个系数（只需要在初始化时读取一遍即可），然后就是传感器测量到的温度数值和气压数值。下面就是要对读到的气压值和温度值进行计算，获得真实的气压（帕斯卡）和温度（摄氏度）。

实际温度和气压的计算

首先看ms5611出厂时已经确定的6个系数：

- c1，气压敏感度，SENS
- c2，气压偏差值，OFF
- c3，气压敏感度的温度系数，简写为TCS
- c4，气压偏差值的温度系数，简写为TCO
- c5，参考温度，Tref
- c6，温度系数，TEMPSENS

上面6个系数为出厂时已经确定，为16位数值。不同的MS5611芯片会不相同，但是同一片芯片是固定的，只需要在初始化时读取一遍，用于后面的计算。

进行D1和D2转化，获得气压数值d1和温度数值d2，为24位数值。

得到了上面8个值之后，就可以计算当前的实际温度和实际气压了。计算过程如下。

Read digital pressure and temperature data

D1	Digital pressure value	unsigned int 32	24	0	16777216	9085466
D2	Digital temperature value	unsigned int 32	24	0	16777216	8569150

Calculate temperature

dT	Difference between actual and reference temperature [2] $dT = D2 - T_{REF} = D2 - C5 * 2^8$	signed int 32	25	-16776960	16777216	2366
TEMP	Actual temperature (-40…85°C with 0.01°C resolution) $TEMP = 20°C + dT * TEMPSENS = 2000 + dT * C6 / 2^{23}$	signed int 32	41	-4000	8500	2007 = 20.07 °C

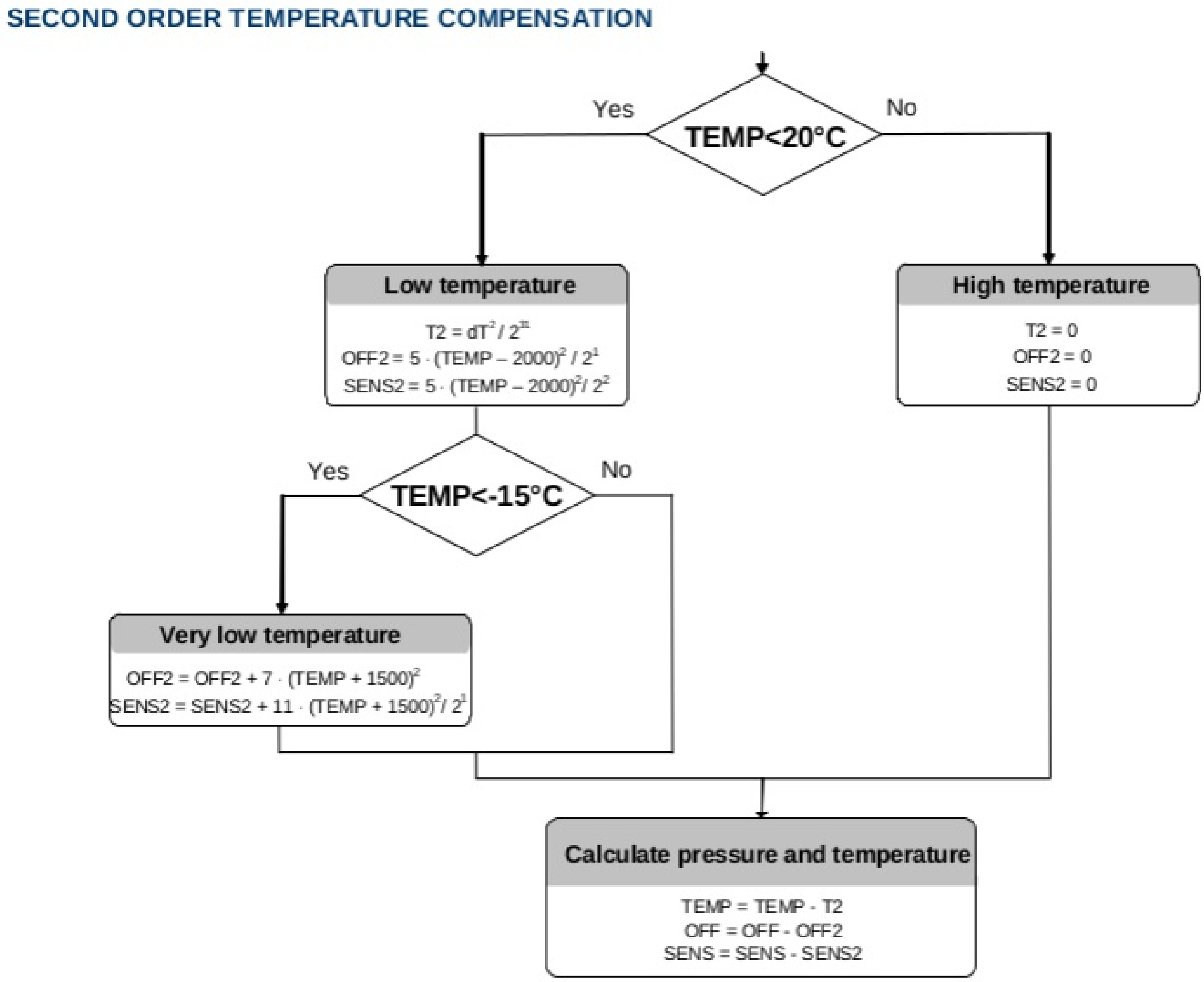
Calculate temperature compensated pressure

OFF	Offset at actual temperature [3] $OFF = OFF_{T1} + TCO * dT = C2 * 2^{16} + (C4 * dT) / 2^7$	signed int 64	41	-8589672450	12884705280	2420281617
SENS	Sensitivity at actual temperature [4] $SENS = SENS_{T1} + TCS * dT = C1 * 2^{15} + (C3 * dT) / 2^8$	signed int 64	41	-4294836225	6442352640	1315097036
P	Temperature compensated pressure (10…1200mbar with 0.01mbar resolution) $P = D1 * SENS - OFF = (D1 * SENS / 2^{21} - OFF) / 2^{15}$	signed int 32	58	1000	120000	100009 = 1000.09 mbar

Display pressure and temperature value

二阶温度补偿

不同温度对大气压强测量有着很大影响，MS5611手册提供有二阶温度补偿的方法。如果温度低于20摄氏度，则进行二阶温度补偿。



扩展-气压计的应用

智能手机

气压传感器首次在智能手机上使用是在Galaxy Nexus上，而之后推出的一些Android旗舰手机里也包含了这一传感器，像GalaxySIII、GalaxyNote2和小米2手机上也都有。

登山测高

对于喜欢登山的人来说，都会非常关心自己所处的高度。海拔高度的测量方法，一般常用的有2种方式，一是通过GPS全球定位系统，二是通过测出大气压，然后根据气压值计算出海拔高度。

导航辅助

现在不少开车人士会用手机来进行导航，不过常常会有人抱怨在高架桥里导航常常会出错 。比如在高架桥上时，GPS说右转，而实际上右边根本没有右转出口，这主要是GPS无法判断你是桥上还是桥下而造成的错误导航。一般高架桥上下两层的高度都会有几米到十几米的距离了，而GPS的误差可能会有几十米，所以发生上面的事情也就可以理解了。

而如果手机里增加一个气压传感器就不一样了，他的精度可以做到1米的误差，这样就可以很好的辅助GPS来测量出所处的高度，错误导航的问题也就容易解决了。

室内定位

由于在室内无法很好的接收GPS信号，所以当使用者进入一幢很厚的楼宇时，内置感应器可能会失去卫星的信号，所以无法识别用户的地理位置，并且无法感知垂直高度。而如果手机加上气压传感器再配合加速计、陀螺仪等技术就可以做到精准的室内定位。这样以后你在商场购物时，就可以通过手机定位来告诉你你想购买的产品在商场的那个位置，哪一层楼。

渔具

另外气压传感器还可以为钓鱼爱好者提供相关信息（鱼在水中分层及活跃性与大气压相关）或天气预报等功能。

不过目前气压传感器还处于一个被忽略的状态，气压传感器要想被更多人了解和使用还需要一些相关技术的成熟和普及，以及更多的开发者针对这一传感器推出更多的应用和相关功能。

博文精选

这里精选了部分发表在Crazepony博客上的文章，可以从更多的层面了解我们的开发历程和其中的故事。



Carmack平方根函数经典案例

> 作者： nieyong
>
> 编者注：在Crazepony飞控代码姿态解算融合中，有一个只有短短不到十行的函数，求解平方根的倒数。该函数是著名的Carmack平方根求解算法，拥有极高的运算效率，堪称经典。本文就介绍该函数背后的故事，由Craze团队-nieyong整理编辑。

首先粘贴Crazepony飞控代码 `IMUSO3.c` 文件中的该函数，求一个浮点数的平方根的倒数。

```
static float invSqrt(float number)
{
    volatile long i;
    volatile float x, y;
    volatile const float f = 1.5F;

    x = number * 0.5F;
    y = number;
    i = * (( long * ) &y);
    i = 0x5f375a86 - ( i >> 1 );
    y = * (( float * ) &i);
    y = y * ( f - ( x * y * y ) );
    return y;
}
```

Quake-III Arena（雷神之锤3）是90年代的经典游戏之一。该系列的游戏不但画面和内容不错，而且即使计算机配置低，也能极其流畅地运行。这要归功于它3D引擎的开发者约翰-卡马克（John Carmack）。事实上早在90年代初DOS时代，只要能在PC上搞个小动画都能让人惊叹一番的时候，John Carmack就推出了石破天惊的Castle Wolfstein, 然后再接再励，doom, doom-II, Quake...每次都把3D技术推到极致。他的3D引擎代码极度高效，几乎是在压榨PC机的每条运算指令。当初微软的Direct3D也得听取他的意见，修改了不少API。

最近，Quake的开发商ID SOFTWARE遵守GPL协议，公开了Quake-III的原代码，让世人有幸目睹Carmack传奇的3D引擎的原码。

我们知道，越底层的函数，调用越频繁。3D引擎归根到底还是数学运算。那么找到最底层的数学运算函数（在 `game/code/q_math.c` ），必然是精心编写的。里面有很多有趣的函数，很多都令人惊奇。在 `game/code/q_math.c` 里发现了这样一段代码。它的作用是将一个数开平方并取倒，经测试这段代码比 `(float)(1.0/sqrt(x))` 快4倍。

```
float Q_rsqrt(float number)
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = * ( long * ) &y;   // evil floating point bit level hacking
    i = 0x5f3759df - ( i >> 1 ); // what the fuck?
    y = * ( float * ) &i;
    y = y * ( threehalfs - ( x2 * y * y ) ); // 1st iteration
    // y   = y * ( threehalfs - ( x2 * y * y ) ); // 2nd iteration, this can be removed

    #ifndef Q3_VM
    #ifdef __linux__
         assert( !isnan(y) ); // bk010122 - FPE?
    #endif
    #endif
    return y;
}
```

函数返回平方根的倒数，即 `1/sqrt(x)` ，这个函数在图像处理中比标准函数 `sqrt(x)` 更有用。

注意到这个函数只用了一次叠代。编译，实验，这个函数不仅工作的很好，而且比标准的 `sqrt()` 函数快4倍！要知道，编译器自带的函数，可是经过严格仔细的汇编优化的啊！这个简洁的函数，最核心，也是最让人费解的，就是标注了“what the fuck?”的一句

> i = 0x5f3759df - (i >> 1);

再加上 `y = y * ( threehalfs - ( x2 * y * y ) );` ，两句话就完成了开方运算！而且注意到，核心那句是定点移位运算，速度极快！特别在很多没有乘法指令的RISC结构CPU上，这样做是极其高效的。

算法的原理其实不复杂,就是牛顿迭代法,用 `x-f(x)/f'(x)` 来不断的逼近 `f(x)=a` 的根。

一般的求平方根都是这么循环迭代算的，但是卡马克真正牛B的地方是他选择了一个神秘的常数0x5f3759df来计算那个猜测值，就是加注释的那一行，那一行算出的值非常接近1/sqrt(n)，这样我们只需要2次牛顿迭代就可以达到我们所需要的精度。好吧如果这个还不算厉害，接着看。

普渡大学的数学家Chris Lomont看了以后觉得有趣，决定要研究一下卡马克弄出来的这个猜测值有什么奥秘。Lomont也是个牛人，在精心研究之后从理论上也推导出一个最佳猜测值，和卡马克的数字非常接近, 0x5f37642f。Lomont计算出结果以后非常满意，于是拿自己计算出的最佳猜测值和卡马克的神秘数字做比赛，看看谁的数字能够更快更精确的求得平方根。结果是卡马克赢了...谁也不知道卡马克是怎么找到这个数字的。

最后Lomont怒了，采用暴力方法一个数字一个数字试过来，终于找到一个比卡马克数字要好上那么一丁点的数字，虽然实际上这两个数字所产生的结果非常近似，这个暴力得出的数字是0x5f375a86。Crazepony飞控代码中，就是使用的这个值。

> Lomont为此写下一篇论文，"Fast Inverse Square Root"。论文下载地址：http://www.math.purdue.edu/~clomont/Math/Papers/2003/InvSqrt.pdf或者http://www.matrix67.com/data/InvSqrt.pdf。参考

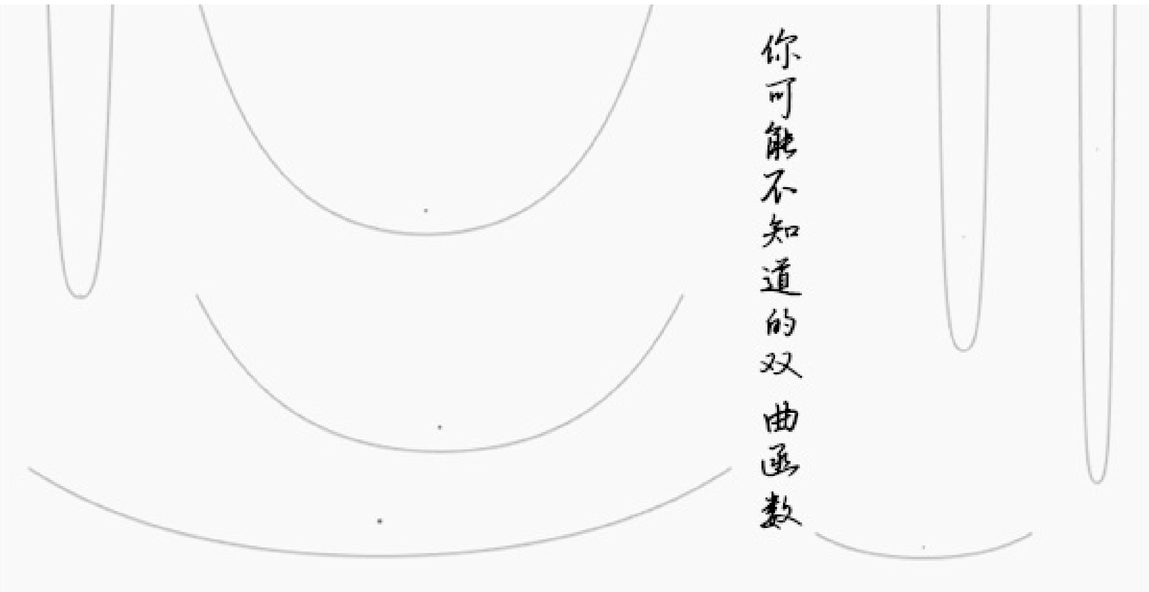
大家可以尝试在PC机、51、AVR、430、ARM、上面编译并实验，惊讶一下它的工作效率。

有则新闻，大意是说Ryszard Sommefeldt很久以前看到这段code，他一看之下惊为天人，想要拜见这位前辈高人，但是一路追寻下去却一直找不到人。在大家追寻的过程中，有人提到一份叫做MIT HACKMEM的文件，这是1970年代的MIT强者们做的一些笔记 (hack memo)，大部份是algorithm，有些 code 是PDP-10 asm写的，另外有少数是C code (有人整理了一份列表)。

这个函数之所以重要，是因为求平方根倒数这个动作在3D运算（向量运算的部份）里面常常会用到，如果你用最原始的sqrt()然后再倒数的话，速度比上面的这个版本大概慢了四倍吧。当然，在飞控多旋翼的姿态解算中，同样涉及到很多向量运算，需要求平方根倒数，所以我们Crazepony选用了这样一个经典的函数。数学，算法是软件的根基，无论是对于游戏**3D**引擎还是无人机多旋翼飞控，都是相通的。**Carmack**平方根函数就是很好的证明。

双曲函数和双曲线的故事

> 作者： nieyong
>
> 编者注：Crazepony微型四轴飞行器机身轮廓使用了余弦双曲线，有着更加优美的流线型和科技感。这里总结了一些关于双曲函数，双曲线的故事和特征，让你更加了解这个优雅而重要的数学函数。下面的文章来自知乎专栏文章《可能是最好的讲解双曲函数的文章》，稍有修改和删减。



一、发展历史

双曲函数的起源是悬链线，首先提出悬链线形状问题的人是达芬奇。他绘制《抱银貂的女人》时曾仔细思索女人脖子上的黑色项链的形状，遗憾的是他没有得到答案就去世了。

时隔170年之久，著名的雅各布·伯努利在一篇论文中又提出了这个问题，并且试图去证明这是一条抛物线。事实上，在他之前的伽利略和吉拉尔都猜测链条的曲线是抛物线。

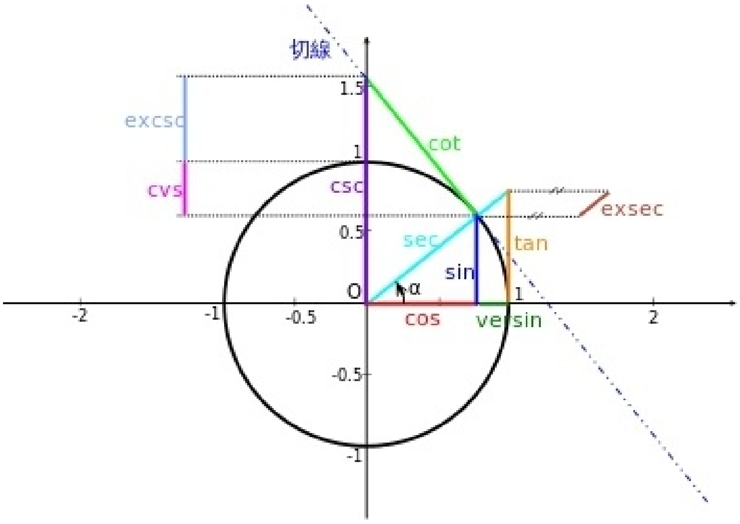
一年之后，雅各布的证明毫无进展（废话，证明错的东西怎么会有进展）。而他的弟弟约翰·伯努利却解出了正确答案，同一时期的莱布尼茨也正确的给出了悬链线的方程。他们的方法都是利用微积分，根据物理规律给出悬链线的二次微分方程然后再求解。

18世纪，约翰·兰伯特开始研究这个函数，首次将双曲函数引入三角学；19世纪中后期，奥古斯都·德·摩根将圆三角学扩展到了双曲线，威廉·克利福德则使用双曲角参数化单位双曲线。至此，双曲函数在数学上已经占有了举足轻重的地位。

19世纪有一门学科开始了全面发展——复变函数。伴随着欧拉公式的诞生，双曲函数与三角函数这两类看起来截然不同的函数获得了前所未有的统一。

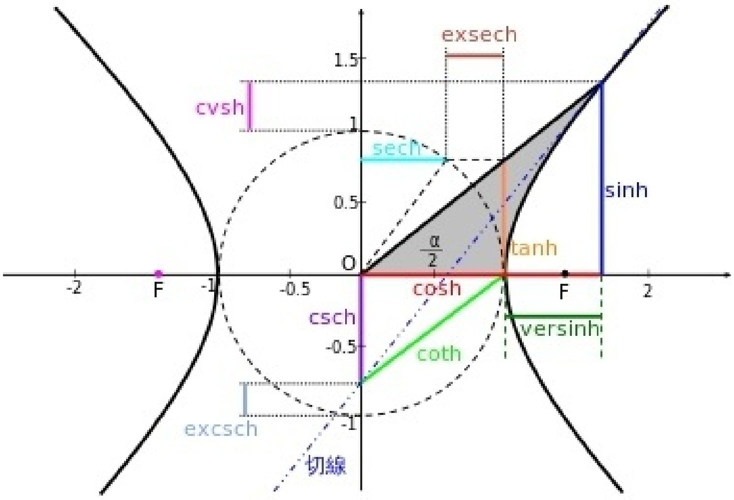
二、函数定义

在讲双曲函数的定义之前，我们先看一看三角函数的定义。如图所示：



在实域内，三角函数的值是通过单位圆和角终边上三角函数线的长度定义的。当然这个「长度」是有正负的。

同理，双曲函数的值也是通过双曲线和角终边上的双曲函数线的长度定义的。如图：

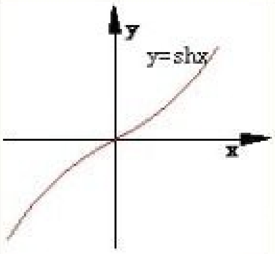
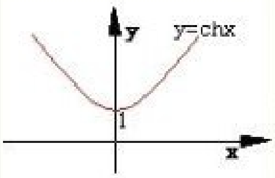
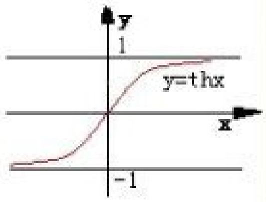


具体的定义为

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

三、函数性质

和对应的三角函数性质十分类似，但又有一定的区别。

函数的名称	函数的表达式	函数的图形	函数的性质
双曲正弦	$shx=\dfrac{e^x-e^{-x}}{2}$		a)：其定义域为：(-∞,+∞)； b)：是奇函数； c)：在定义域内是单调增
双曲余弦	$chx=\dfrac{e^x+e^{-x}}{2}$		a)：其定义域为：(-∞,+∞)； b)：是偶函数； c)：其图像过点(0,1)；
双曲正切	$thx=\dfrac{e^x-e^{-x}}{e^x+e^{-x}}$		a)：其定义域为：(-∞,+∞)； b)：是奇函数； c)：其图形夹在水平直线 y=1 及 y=-1 之间；在定域内单调增；

四、恒等式

双曲函数恒等式一定要结合着三角函数恒等式一起看，真的是太像了：

圆函数（三角函数）

1.基本性质：

$$\tan x=\frac{\sin x}{\cos x},\cot x=\frac{\cos x}{\sin x}$$

$$\sec x=\frac{1}{\cos x},\csc x=\frac{1}{\sin x}$$

$$\tan x\cot x=1$$
$$\sin x\csc x=1$$
$$\sec x\cos x=1$$

$$\sin^2 x+\cos^2 x=1$$

$$1+\tan^2 x=\sec^2 x,1+\cot^2 x=\csc^2 x$$

2.奇偶性：

$$\sin(-x)=-\sin x$$

$$\cos(-x)=\cos x$$

$$\tan(-x)=-\tan x$$

3.两角和差公式

$$\sin(x\pm y)=\sin x\cos y\pm\cos x\sin y$$

$$\cos(x\pm y)=\cos x\cos y\mp\sin x\sin y$$

$$\tan(x\pm y)=\frac{\tan x\pm\tan y}{1\mp\tan x\tan y}$$

4.二倍角公式

$$\sin 2x=2\sin x\cos x$$

$$\begin{aligned}\cos 2x&=\cos^2 x-\sin^2 x\\&=2\cos^2 x-1\\&=1-2\sin^2 x\end{aligned}$$

$$\tan 2x=\frac{2\tan x}{1-\tan^2 x}$$

双曲函数

1.基本性质：

$$\operatorname{th} x=\frac{\operatorname{sh} x}{\operatorname{ch} x},\operatorname{cth} x=\frac{\operatorname{ch} x}{\operatorname{sh} x}$$

$$\operatorname{sech} x=\frac{1}{\operatorname{ch} x},\operatorname{csch} x=\frac{1}{\operatorname{sh} x}$$

$$\operatorname{th} x\operatorname{cth} x=1$$
$$\operatorname{sh} x\operatorname{csch} x=1$$
$$\operatorname{sech} x\operatorname{ch} x=1$$

$$\operatorname{ch}^2 x-\operatorname{sh}^2 x=1$$

$$1-\operatorname{th}^2 x=\operatorname{sech}^2 x,1-\operatorname{cth}^2 x=-\operatorname{csch}^2 x$$

2.奇偶性：

$$\operatorname{sh}(-x)=-\operatorname{sh} x$$

$$\operatorname{ch}(-x)=\operatorname{ch} x$$

$$\operatorname{th}(-x)=-\operatorname{th} x$$

3.两角和差公式

$$\operatorname{sh}(x\pm y)=\operatorname{sh} x\operatorname{ch} y\pm\operatorname{ch} x\operatorname{sh} y$$

$$\operatorname{ch}(x\pm y)=\operatorname{ch} x\operatorname{ch} y\pm\operatorname{sh} x\operatorname{sh} y$$

$$\operatorname{th}(x\pm y)=\frac{\operatorname{th} x\pm\operatorname{th} y}{1\pm\operatorname{th} x\operatorname{th} y}$$

4.二倍角公式

$$\operatorname{sh} 2x=2\operatorname{sh} x\operatorname{ch} x$$

$$\begin{aligned}\operatorname{ch} 2x&=\operatorname{ch}^2 x+\operatorname{sh}^2 x\\&=2\operatorname{ch}^2 x-1\\&=1+2\operatorname{sh}^2 x\end{aligned}$$

$$\operatorname{th} 2x=\frac{2\operatorname{th} x}{1+\operatorname{th}^2 x}$$

五、欧拉公式

欧拉公式是复变函数里几乎最重要的一个公式，它揭示了三角函数和指数函数之间的内在联系，从形式上也十分简洁优美：

$$e^{ix}=\cos x+i\sin x$$

用-x替换掉x，得到

$$e^{-ix}=\cos x-i\sin x$$

这样我们可以解出正弦和余弦函数与指数函数的关系式：

$$\cos x=\frac{e^{ix}+e^{-ix}}{2}$$

再把双曲函数拉过来看看

$$\sinh x=\frac{e^{x}-e^{-x}}{2}$$

是不是非常接近了呢？很容易看出它们之间存在这样的关系：

$$\cos x=\cosh(ix),$$
$$\sin x=-i\sinh(ix)$$

六、复域统一

先研究一下三角函数和双曲函数的级数展开。

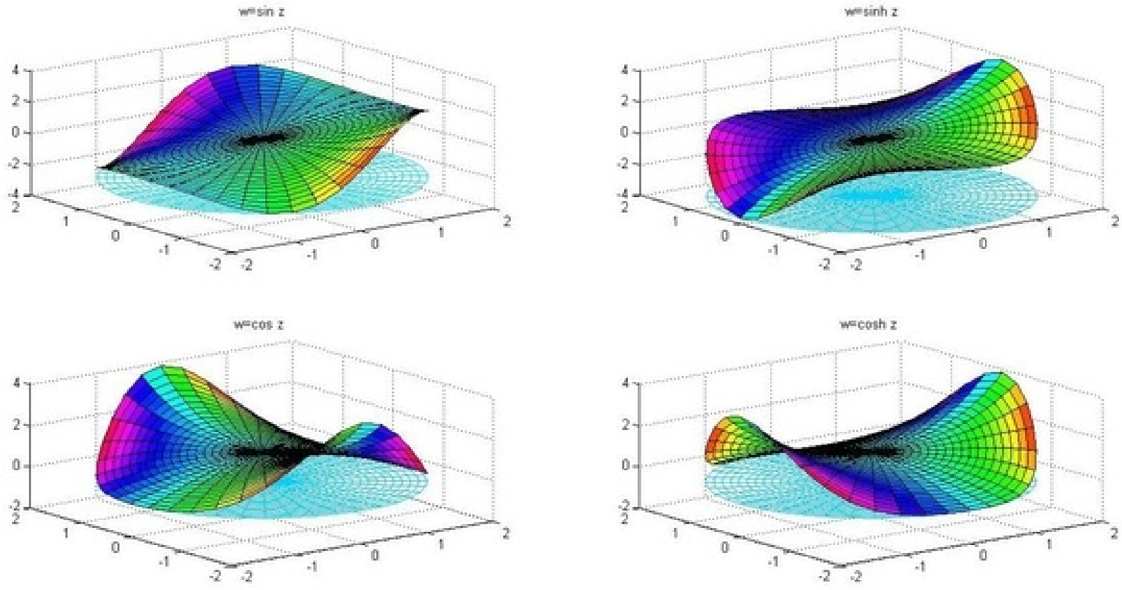
$$\sin x=\sum_{n=0}^{\infty}\frac{(-1)^n x^{2n+1}}{(2n+1)!}$$

双曲函数和三角函数的区别仅仅在于是否有-1的幂这一项，双曲函数就是将三角函数改为非交错级数。正是由于其无比相似的级数展开，才使得它们具有十分相似的性质。

我们说了这么多，两类函数似乎各种相似却还是不一样。那么三角函数和双曲函数的关系到底是什么呢？

在复域上，它们的形状其实是一样的！

不信？我们画一画图像。



这两个式子说明对应的两个函数仅通过旋转（对于复变函数，乘i就相当于逆时针旋转90°）即可重合。

对了，大家都知道三角函数的周期是2pi，那么大家猜猜双曲函数的周期是多少？没错，是2ipi！

七、映射关系

需具备复变函数基础，该部分省略……

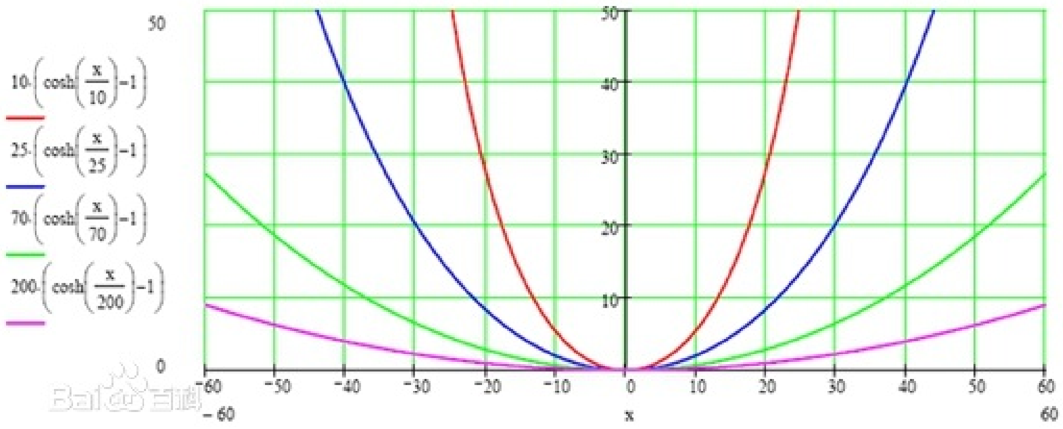
八、应用范围

1.悬链线

悬链线的方程是双曲余弦函数，这个在文章开头已经介绍过。而悬索桥、双曲拱桥、架空电缆等都用到了悬链线的原理。在工程上，定义a为悬链线系数，而把悬链的方程记为

$$y = a(\cosh\frac{x}{a} - 1)$$

给应用带来很大的方便，如图：



2.平行直导线单位长度电容

下面部分省略……

Crazepony鞭状天线室外飞行记

> 作者： nieyong

进入2015年，Crazepony终于迎来了5.1版本。为了这一版本，Craze团队投入了很大的精力，在操作性能，结构组装上都有了不少的改进。昨天我们遥控换上了带有PA的鞭状天线，再次来到开放制造空间，体验了一把室外飞行。

首先是Techspace的Logo特写，这是一个深圳创客聚集的福地。Crazepony也多次来到这里，并且认识了不少这里的小伙伴。



正在静静的充电中。这时候就会问到，充一次电，能够飞多久呢？回答是：我们只能够飞4-5分钟。这已经是Crazepony的350mAh电池的极限了。当然，你可以自己配备一块500mAh的电池。很多朋友会说，我买的司马X5C这样的玩具飞机，能够飞更久呢，为什么你们只能够4分钟。你仔细看哦，X5C这样的玩具飞机，可是有减速齿轮的。减速齿轮降低了电机的KV值，而原则上低KV值配大桨叶，空气动力效率会高，同样的电池，飞得更久。

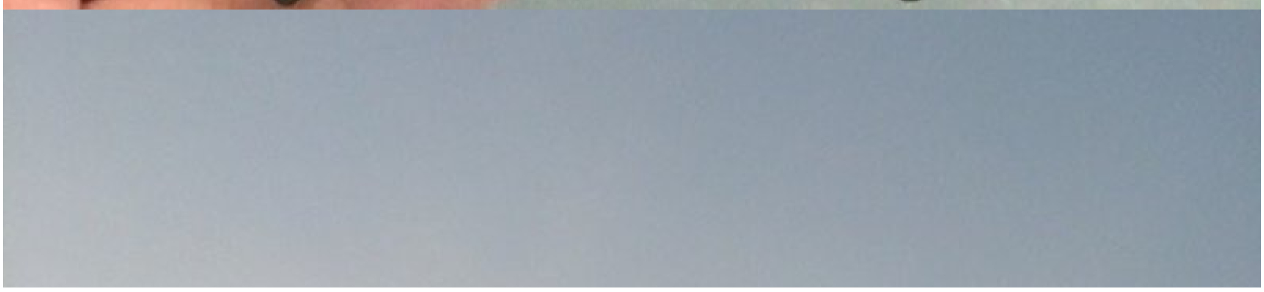


室外飞行靓影。遥控器上装上了鞭状天线，室外飞行再也不怕飞丢了。我们测试可以飞80米，这个时候你基本上看不到飞机了。使用无头模式（carefree）飞行，定是极好的。

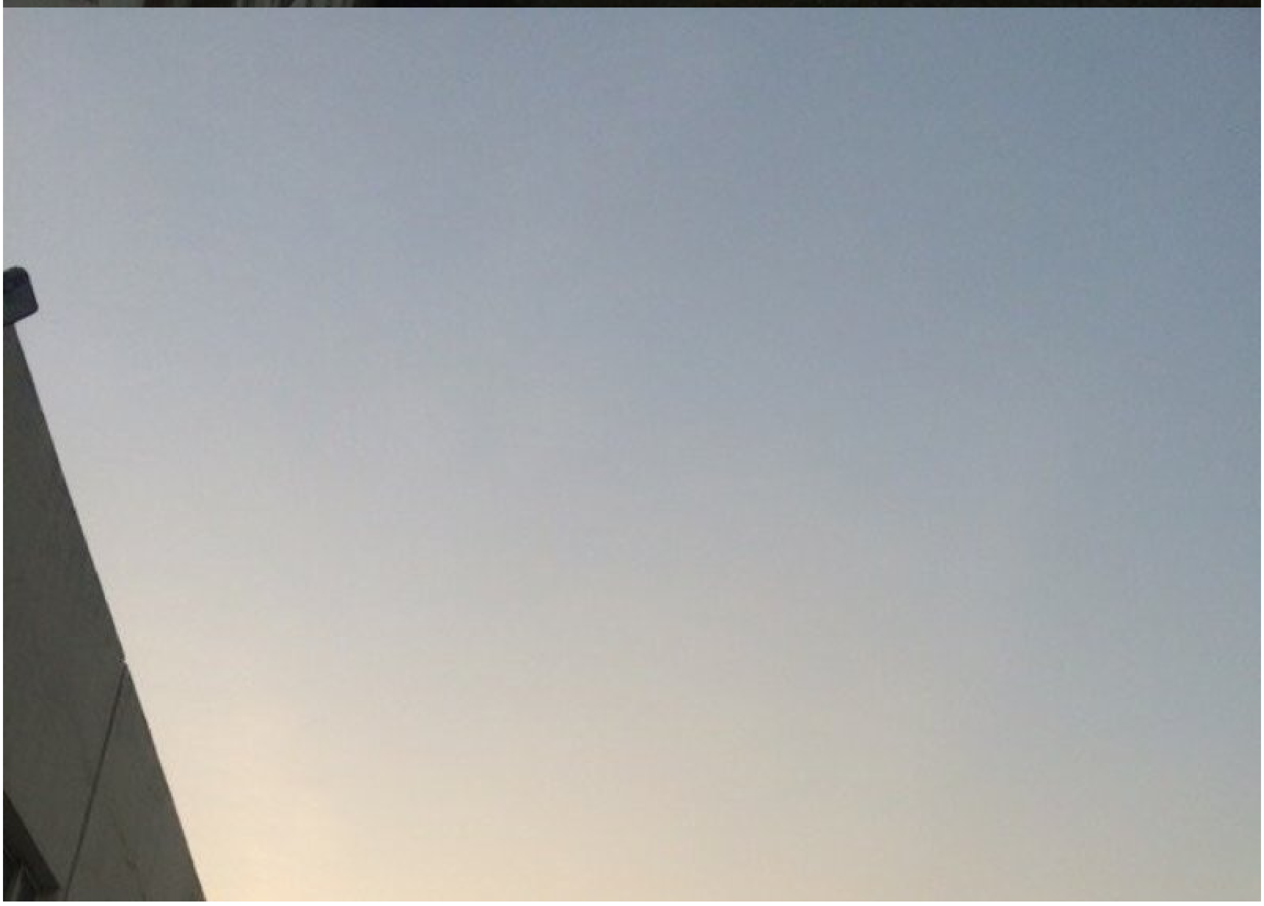


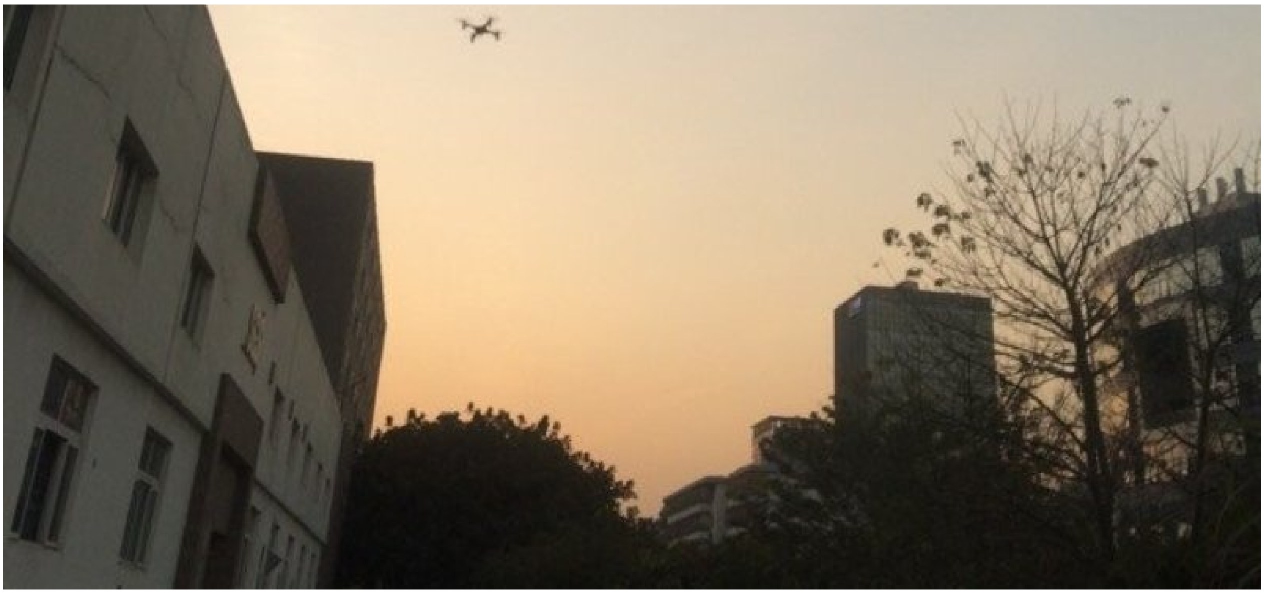
加PA鞭状天线的遥控器及飞远了的Crazepony。遥控器上的2.4G部分是以模块形式存在的，直接插上鞭状天线模块就可以使用，无需修改代码。我们默认配置的不带PA的2.4G模块，可控范围是10米，适合在室内飞行。


Mode
PCB Vber5.1

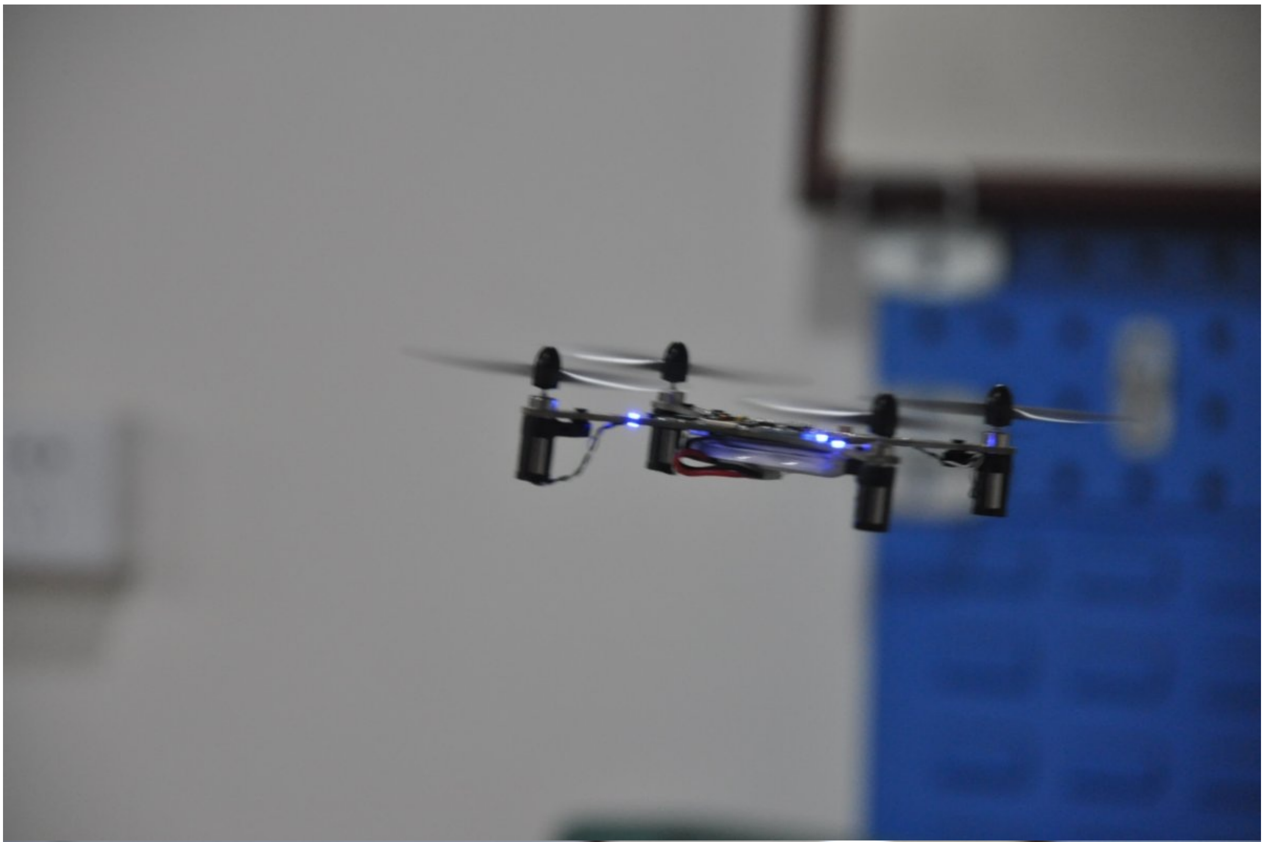








再放几张室内飞行的靓照。在最新版本Crazepony上实现的无头飞行模式，气压计Z轴定高悬停模式，让新手操控变得简单。





CrazePony与计算机视觉

> 作者： Ziv.Lin

大家好,我是CrazePony的视觉攻城狮,Ziv.Lin,我擅长的部分是计算机视觉和嵌入式编程,非常荣幸地能够加入CrazePony团队~

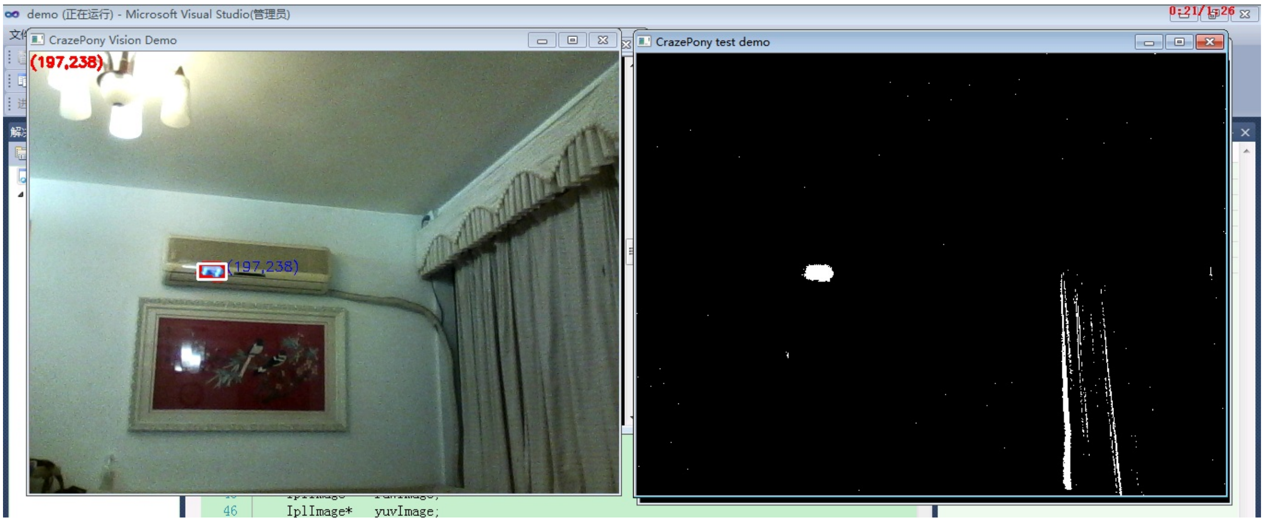
众所周知,一个飞行器想要做到悬停的话，必须引入一个位置环进行反馈，也就是说需要获知无人机的空间绝对坐标，目前的无人机普遍采用GPS（Global Position System）以获得其位置，但是GPS信号存在容易受干扰（甚至是欺骗，伊朗截获美国的RQ170就是采用GPS欺骗的方法）、在室内没有信号等无法克服的弊端，如今，业界正在寻求一种解决方案，以克服GPS的这些先天性的缺点，也就是我今天想要说的计算机视觉(^_^)

对于类似CrazePony这些微型飞行器来说，载重相当有限，不大可能机载GPS系统（因为如果想要获得较低的定位误差,必须使用介电陶瓷的天线，其重量较重）,想要实现悬停等动作,比较可行的方法就是引入视觉定位系统,如下图所示.

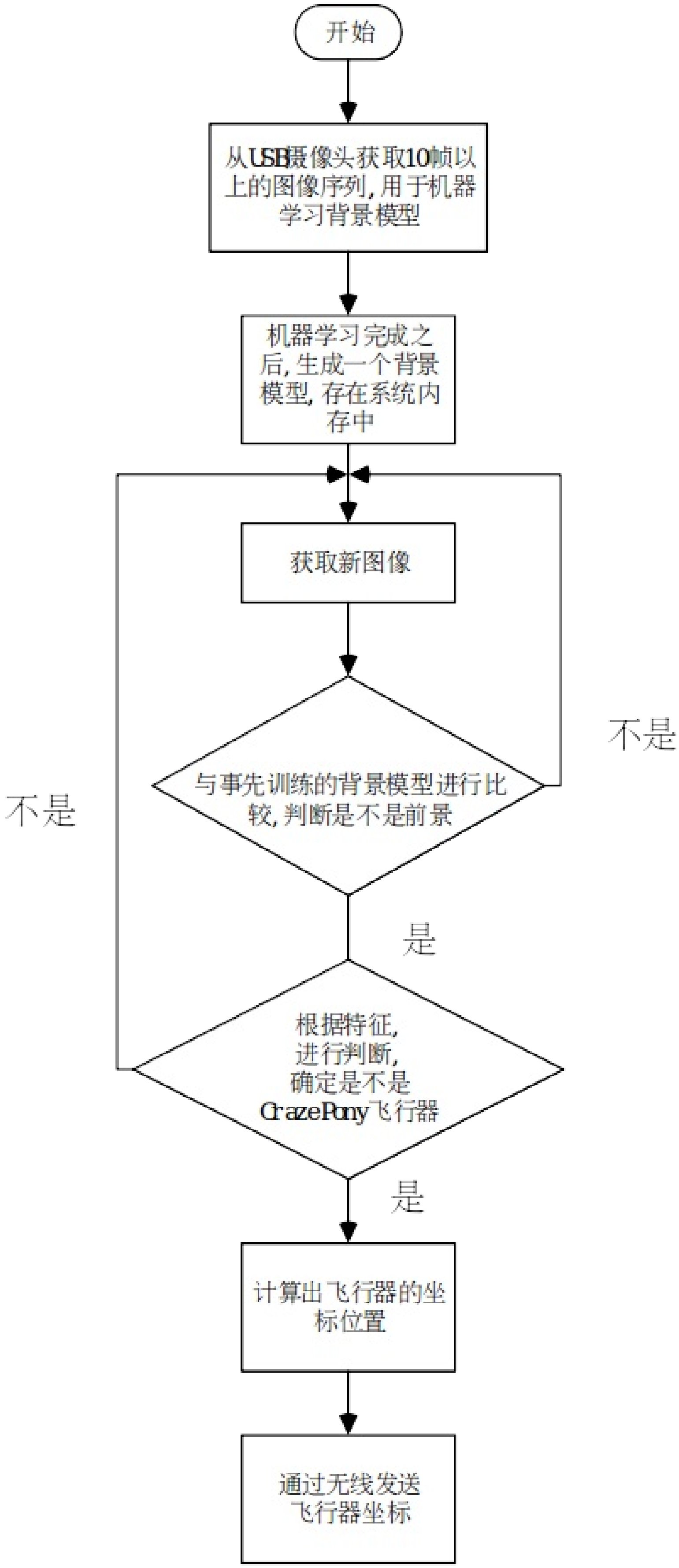


CrazePonzy的计算机视觉定位我打算分为三步走:

第一步:就是实现四轴飞行器的视觉跟踪,现在已经初步完成~



这个视觉定位系统视频已经上传至CrazePony的首页（也附加在本文最后），这个是基于OpenCV库开发的一个定位系统,其算法流程图如下所示:



第二步:就是实现飞行器的航线飞行,现在,我们团队正在努力实现中

第三部:把所有的计算机视觉移植到手机平台去,现在我们已经实现OpenCV库在Android上面的移植了,相信总有一天,你可以使用你的手机飞CrazePony了





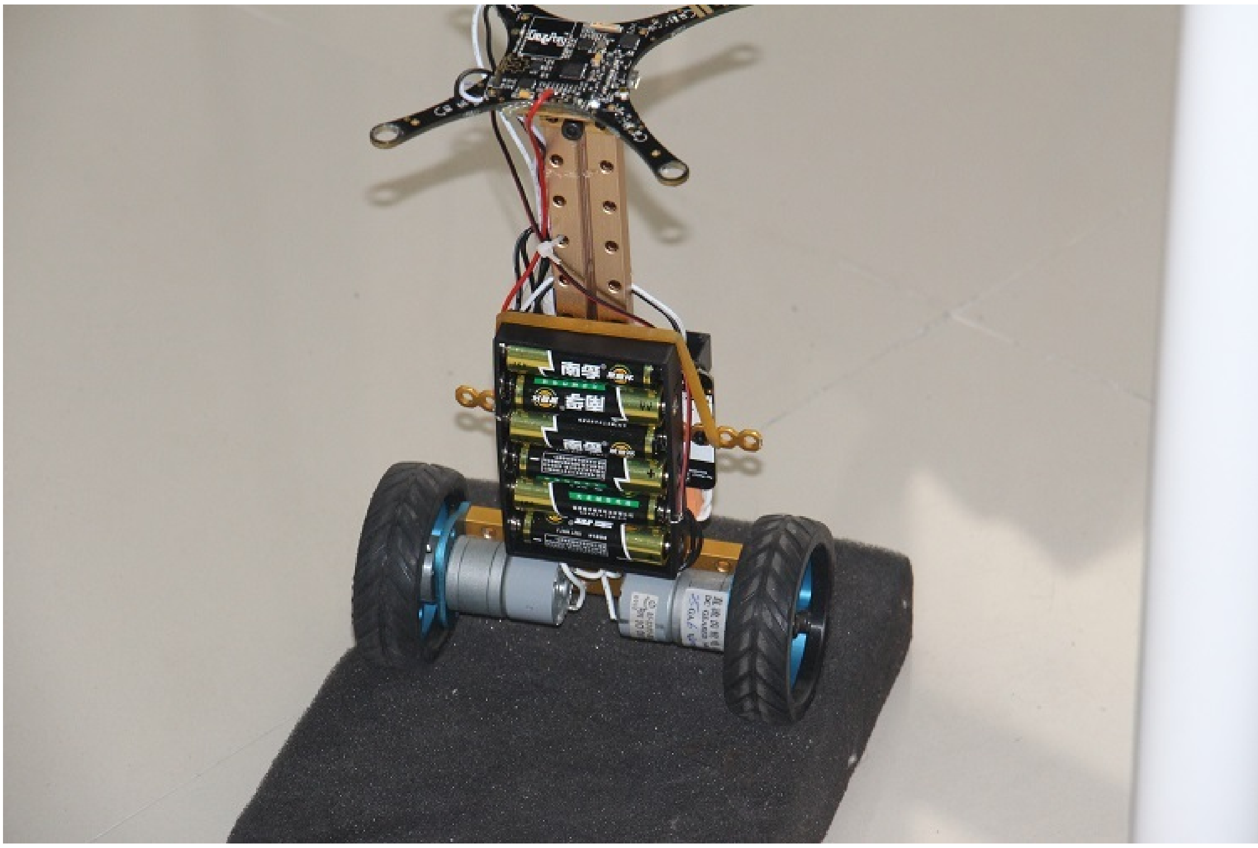
最后附上Crazepony视觉定位系统的优酷视频。

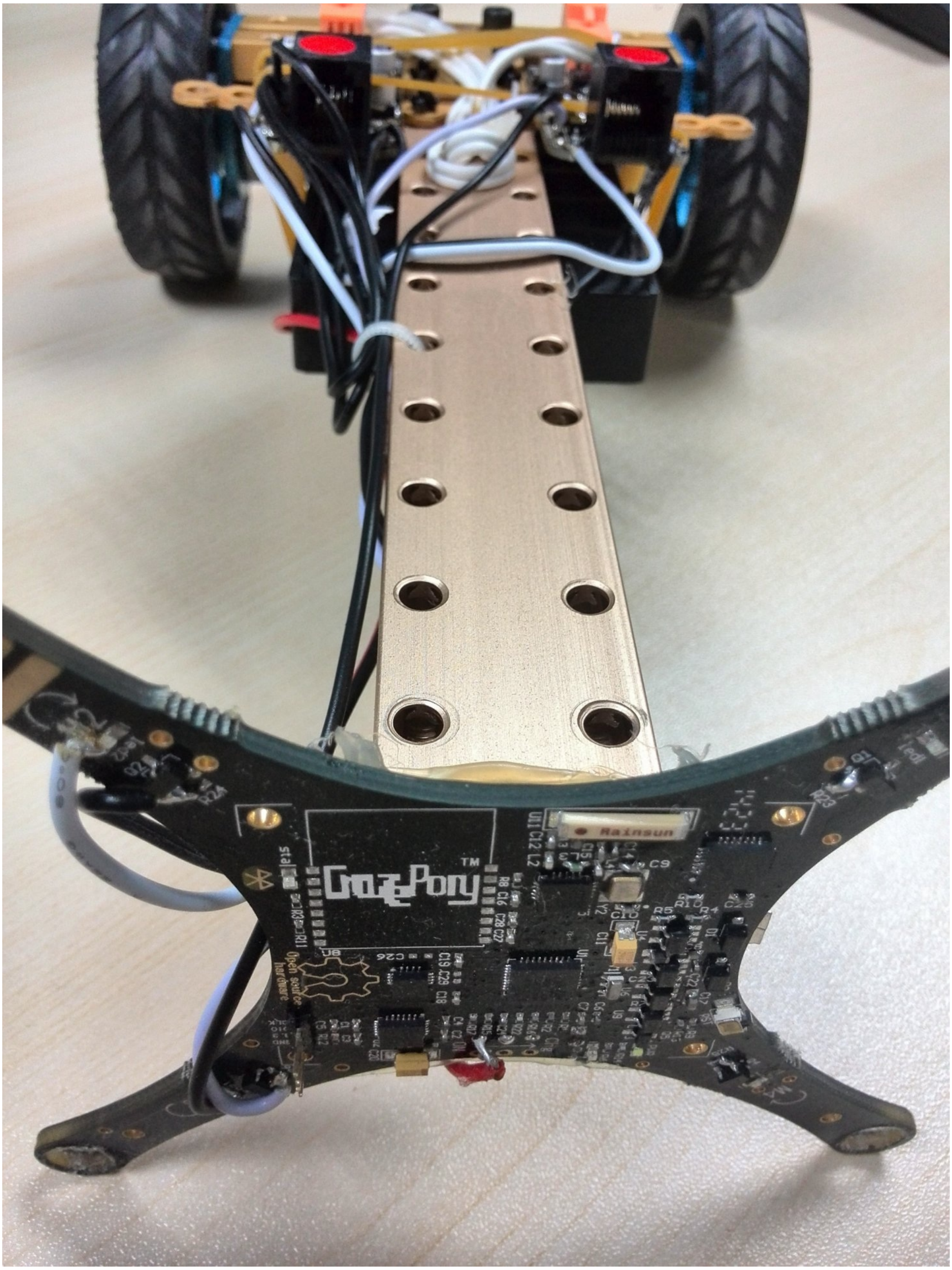
Crazepony自平衡小车

> 作者： nieyong

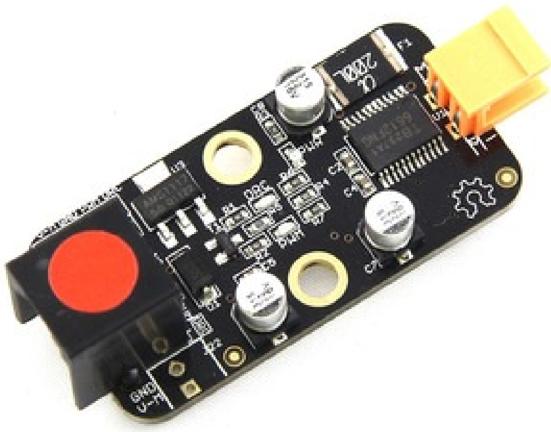
离上次参加makeblock周末马拉松比赛已经快一个月了。一直想介绍我们Craze团队的作品基于**Crazepony**自平衡小车，却迟迟没有动笔。很多人在问为什么主页上关于自平衡小车的链接是空的，大家都等不及了。Crazepony平台不仅仅是一个四轴飞行器，而且还能够很简单的实现自平衡小车。

首先上图。我们使用了makeblock提供的机械构件，包括车轮，电机，以及一些金属连接构件。这里不得不赞一下makeblock的产品，全部是铝合金材质，非常高达上。而且特有的内嵌螺纹槽，给DIY提供了非常大的灵活性。基本上在任何地方都可以上螺丝进行连接。

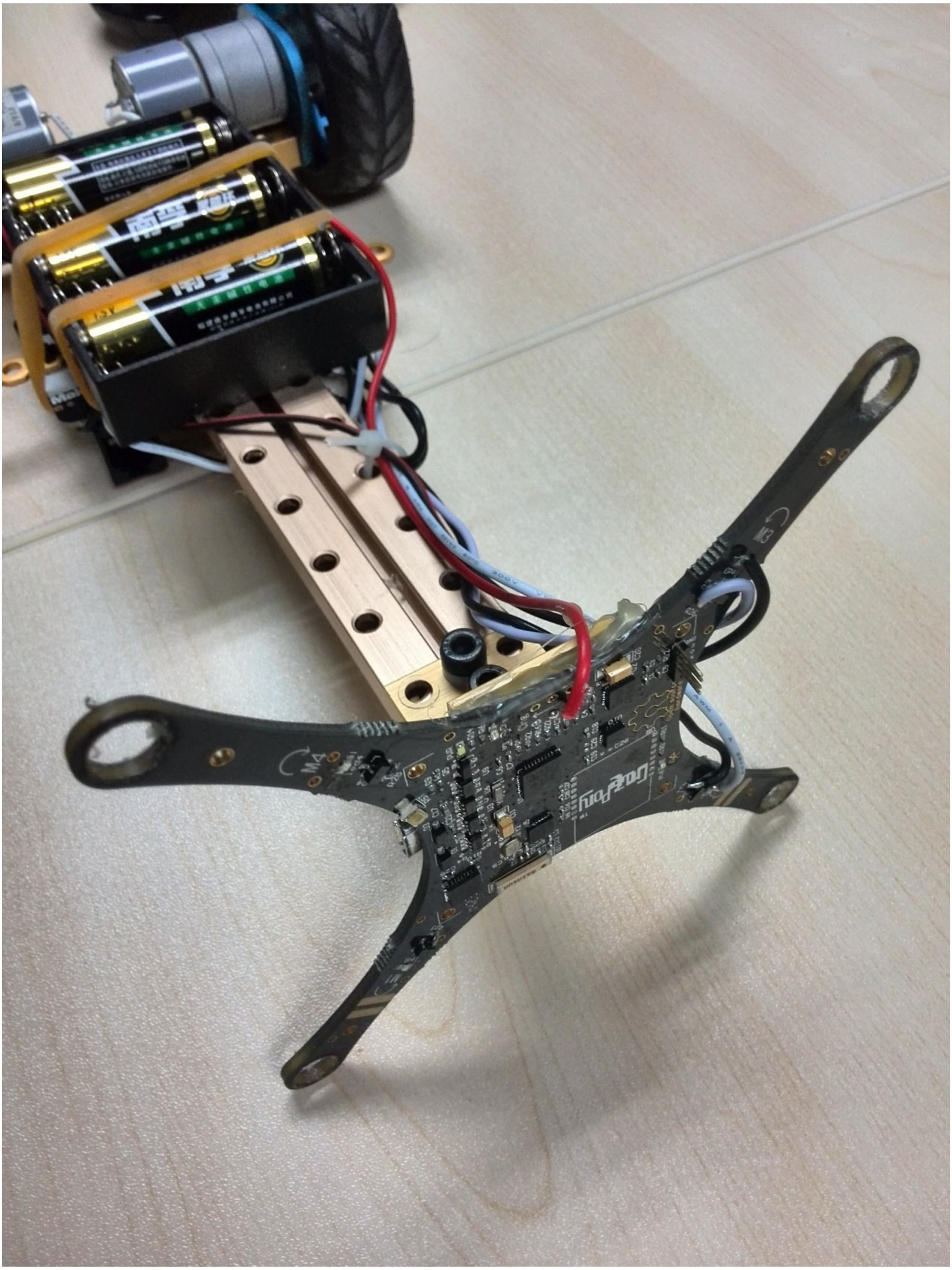


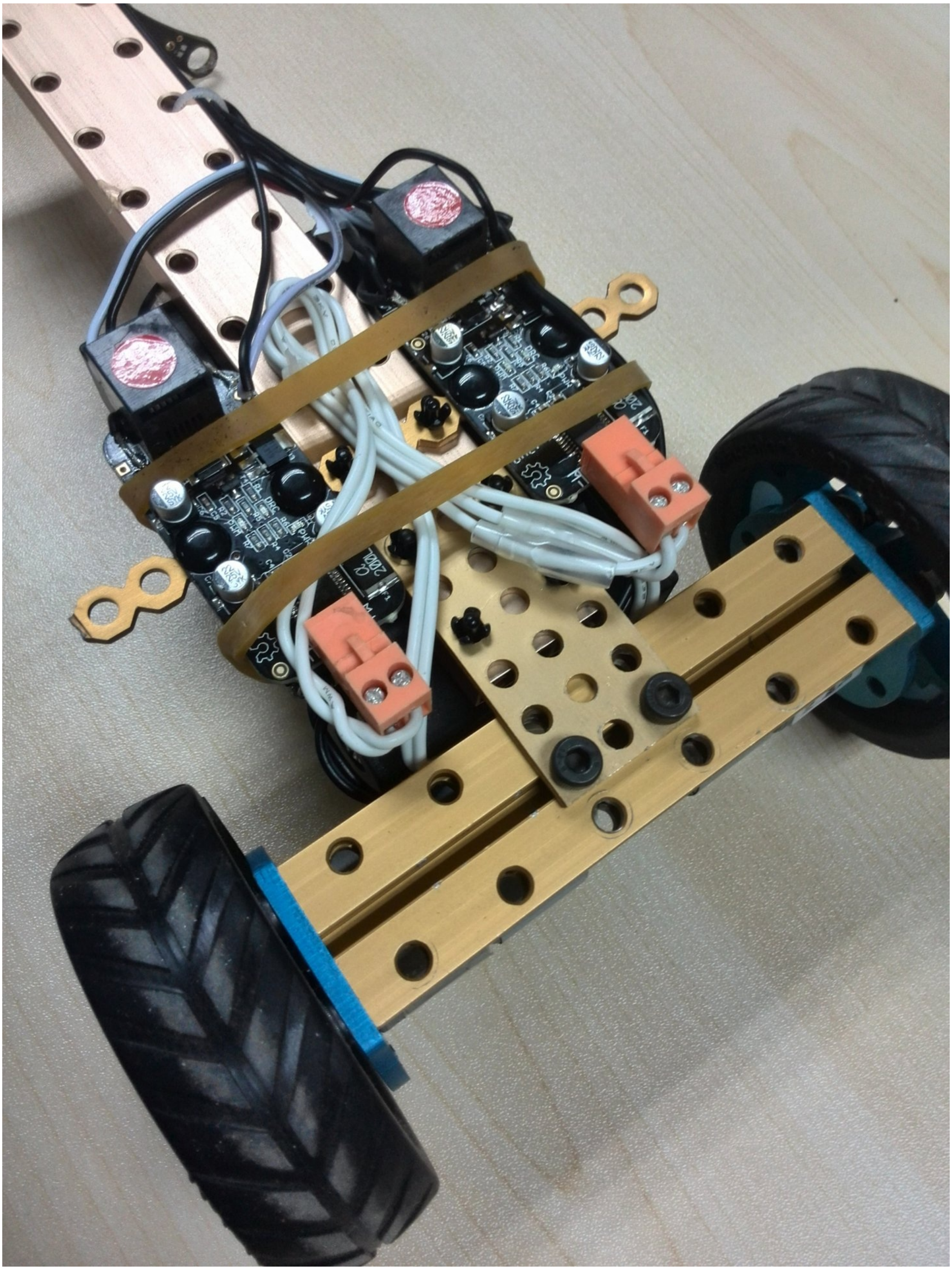


电机驱动电路也使用makeblock提供的Me Motor Driver模块。Me系列模块本来是为降低makeblock电子部分门槛而开发的，使用了RJ-25连接头。这和Crazepony飞控板是不匹配的，所以我们不得不做飞线处理。



最上面就是Crazepony飞控板，使用原来四路电机输出PWM中的两路控制连接到电机驱动模块。





只需要稍稍修改飞控代码（主要是PID参数），我们的自平衡小车就能够站立起来了。在比赛中使用的这个方案是开环控制，没有小车运动速度的反馈作闭环控制，所以我们的自平衡小车并不能够行走自如。我们让它在一个定做的“拳击擂台”里冲撞。它的对手只有一个，那就是它自己，别让自己倒下啦。

最后放上Crazepony在拳击擂台里跌跌撞撞的视频。

DIY四轴飞行器如何入门？

> 作者： 电子匠人
>
> 作者：电子匠人，本文在知乎同标题帖基础上编辑而成。

先回答入门书籍的问题。在淘宝和京东上没见到过DIY四轴的书籍，有一些飞行器的也是建模神马的天书。当时在京东买了一本《空中机器人（四旋翼）专项教育教材》，妈蛋这么偏僻我都找到了！！！！快递到了兴高彩烈的拆包，全书主要讲了空中机器人大赛的重大意义，坑爹呢这是！！！！



四轴有很多种做法，我发一下我知道的两种吧

大四轴

大四轴,无刷电机+电调+飞控板（各种Geek长玩的，买硬件，调参数和算法）



大四轴是最烧钱的了，入手一套大概1000~2000元。我们是DIY，各种零件当然要自己买了，不怕折腾，我们按照如下配置搭建一套大四轴:

- 格氏11.1V2200mA25C锂电 128
- B6充电器 160
- 郎宇A2212电机 62×4
- 螺旋桨8个 40 (需要4个正桨，4个反桨，万一坏了呢剩下备用)
- 天行者20A电调 48×4
- 四轴机架 88
- 飞控板 100 (KK/MWC等等总要玩个开源飞控吧？否则光调参数你都不好意思说出口)
- 天地飞6遥控器 200（6通道 遥控器）

通道就是可以遥控器控制的动作路数，比如遥控器只能控制四轴上下飞，那么就是1个通道。但四轴在控制过程中需要控制的动作路数有：上下、左右、前后、旋转 所以最低得4通道遥控器。如果想以后玩航拍这些就需要更多通道的遥控器了。

算一下总价：**128+160+624+484+88+200+100=1156**

我们把四轴组装起来（会简单的电路焊接就可以了）就可以连接上位机通过电脑调试参数了。



这个界面简直狂拽酷霸吊炸天啊！！

调试主要是PID参数，一般买的飞控简单调调就可以试飞了。日后还可以加GPS神马的玩些高端的定点飞行！

大四轴一定要有一个安全的调试环境，东西要装牢靠，周围不要有行人。还有就是！一旦发生问题千万不要用手去抓四轴!!!!这个是看到我QQ空间上一个好友调四轴受伤的手:



看到有些人调四轴都带护目镜保护眼睛以防螺旋桨断了射出去！！！高速旋转的螺旋桨就像子弹一样，不要以为是玩就没有安全隐患了，绝不要掉以轻心！

小四轴

作为屌丝学生党，大四轴我暂时是玩不起的。于是我就想别的办法，反正都是四轴，小的是不是便宜些又安全些？

小四轴不仅没有电调，连机架都剩了，电机装在电路板上：



没有大四轴飞起来霸气，不能载重。但小四轴有个优点就是想飞就飞，一不受空间限制，二不容易伤人。即使你用手抓螺旋桨，那感觉基本也就是挠痒痒。

小四轴的配置和价格：

- 716空心杯电机 40 （10*4）
- 20C 3.7v 锂电池 15 (与手机电池区别在放电量上）
- 一大堆电子器件 50
- PCB电路板制作 100
- 蓝牙模块 30
- 遥控器用安卓手机 不花钱

计算下 **40+15+50+100+30 = 245**元，相比较大四轴的1156来说少了太多

小四轴的优点：

- 价格便宜
- 从口袋里掏出来就能飞
- 自己设计飞控电路板
- 自己移植编写开源单片机程序

- 手机遥控 连遥控器都省了

小四轴价格便宜，飞行灵活，还能体会到完整的DIY乐趣。下面是我自己 DIY的四轴飞行器（飞控，PC上位机，手机遥控，摇杆）：



我的四轴有个小机架(碳杆+空心杯电机+减速齿轮) 也不贵，几十块钱。

现在看到越来越多的人都开始玩小四轴了，我的四轴就是根据论坛的开源四轴一步步制作的。

分享几个开源四轴的资料：

国外的:

- crazyflie：Bitcraze
- 德国MK：MikroKopter

国内的:

- 匿名四轴：不思带你【从零开始】做四轴！！！(强势整理搬运版)
- 圆点博士：圆点博士微型四轴飞行器

我的博客四轴文章列表 从零做四轴飞行器以及CrazePony四轴飞行器项目。

回答越写越长，不知不觉写了俩小时，最后希望大家早日DIY出自己的四轴飞行器！

《大人的科学》引起的思考

> 作者： nieyong / Jul 07,2014

《大人的科学》是日本的一本科技类的季度杂志。每期介绍一个科学模型，并附送这个科学模型，让读者DIY这个产品，并从中感受到制作的快乐，并学到相应的科学知识。例如第一期是关于蒸汽汽船，第九期是DIY星象仪模型。这本杂志在日本非常受欢迎，取得了很大的成功。

“科学的世界无关年龄，有趣的事物始终有趣”这是该杂志的口号。

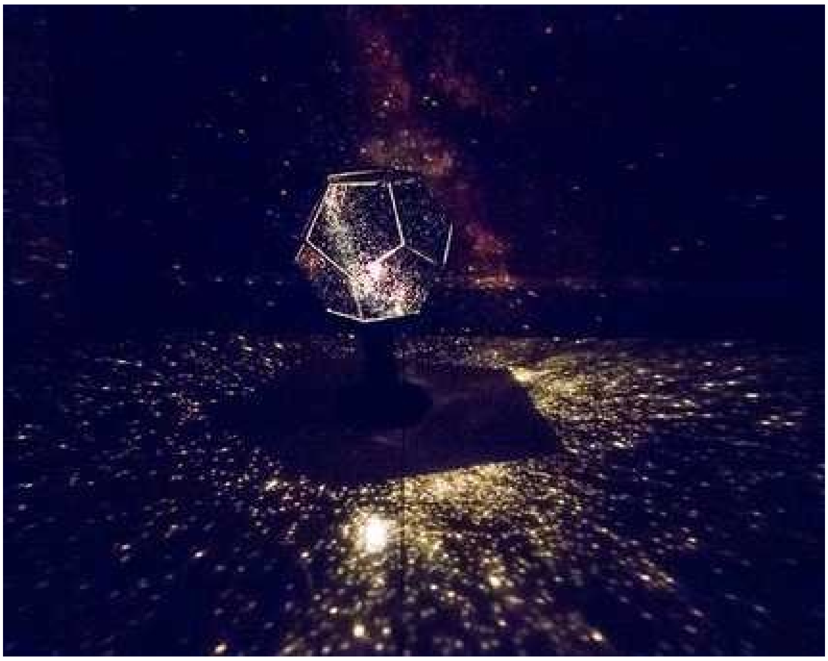




我是通过科学出版社的主编Lee知道这本杂志的。出版社对我们的四轴飞行器感兴趣，就是想借鉴《大人的科学》的模式，做一期四轴飞行器的刊物。

结合自己身处电子行业，对开源硬件/创客的了解，我觉得手工DIY制作套件这个领域，有很大的挖掘前景。这既是一件非常有意义的事情，又能够找到适合自己生存的土壤。《大人的科学》是一个很好的模式和案例，这个市场在中国现在还比较空白。对原来粗糙的手工制作进行重新定位和思考，使用我们现在的理念来包装，运营，销售。

下图就是做好的星象仪效果图。



Crazyflie四轴飞行器

Crazyflie四轴飞行器是一个和Crazepony非常类似的迷你四轴飞行器项目，由瑞士三个工程师设计开发。该项目现在由深圳Seeed Studio公司负责生产和销售。Crazyflie是Crazepony的前辈和学习的对象。

我们研究了Crazyflie的结构/硬件/软件，并且做了相关的笔记和分析。这里我们将笔记分享出来，方便后面人们的研究和学习。



FreeRTOS简介

> 作者：nieyong，CamelGo

FreeRTOS

在嵌入式领域中，嵌入式实时操作系统正得到越来越广泛的应用。采用嵌入式实时操作系统(RTOS)可以更合理、更有效地利用CPU的资源，简化应用软件的设计，缩短系统开发时间，更好地保证系统的实时性和可靠性。

作为一个轻量级的操作系统，FreeRTOS提供的功能包括：任务管理、时间管理、信号量、消息队列、内存管理、记录功能等，可基本满足较小系统的需要。FreeRTOS内核支持优先级调度算法，每个任务可根据重要程度的不同被赋予一定的优先级，CPU总是让处于就绪态的、优先级最高的任务先运行。FreeRT0S内核同时支持轮换调度算法，系统允许不同的任务使用相同的优先级，在没有更高优先级任务就绪的情况下，同一优先级的任务共享CPU的使用时间。

FreeRTOS的内核可根据用户需要设置为可剥夺型内核或不可剥夺型内核。当FreeRTOS被设置为可剥夺型内核时，处于就绪态的高优先级任务能剥夺低优先级任务的CPU使用权，这样可保证系统满足实时性的要求；当FreeRTOS被设置为不可剥夺型内核时，处于就绪态的高优先级任务只有等当前运行任务主动释放CPU的使用权后才能获得运行，这样可提高CPU的运行效率.

Crazepony-II的FreeRTOS移植

目前稳定的裸机程序已暴露出很多弊端,比如CPU资源浪费问题,处理任务单一,只能是中断嵌套或者CPU轮询机制去处理各类事件,CPU空挡期远大于实际利用期.

所以,目前我们正专人负责移植FreeRTOS系统,在系统层面做到兼容Bitcraze的crazefile固件,在这期间肯定会遇到很多问题,这都需要大家在实际体验过程中发现并解决问题,在Crazepony这个平台更加稳定,更加成熟

介绍几个常用的宏的作用

> 作者：韦晟敢

在Crazyflie的固件代码中，宏一般都定义在文件Makefile中。使用 `CFLAGS += -D` 的方式添加。

```
# Flag that can be added to config.mk
CFLAGS += -DUSE_UART_CRTP        # Set CRTP link to UART
CFLAGS += -DUSE_ESKYLINK         # Set CRTP link to E-SKY receiver
CFLAGS += -DENABLE_UART          # To enable the uart
CFLAGS += -DDEBUG_PRINT_ON_UART  # Redirect the console output to the UART
```

ENABLE_UART

启用串口打印功能。要想调试期间把信息打印到串口，还需要加上宏：DEBUG_PRINT_ON_UART

USE_UART_CRTP

crtp模块为：Crazy Realtime Transfer Protocol stack，是crazy flie的通信协议栈。通信链路可以2.4G 遥控信息，可以使用USE_ESKYLINK，也可以使用串口。若要使用蓝牙模块作为通信链路，应该开启此宏。

DEBUG_PRINT_ON_UART

使用串口调试。在平衡算法当中进行调试的时候，串口打印显得尤为重要，通过此宏可以开启串口调试。注意：开启串口调试的时候不能开启宏USE_UART_CRTP，串口作为通信链接。

BOARD_2

此宏在drivers/i2c_gpio.c文件中。因为版本2与版本3的i2c部分引脚有区别，去掉此宏定义适用于第三版的硬件

系统流程框架

> 作者：nieyong

介绍系统的初始化流程，以及系统任务之间的关系。

系统任务

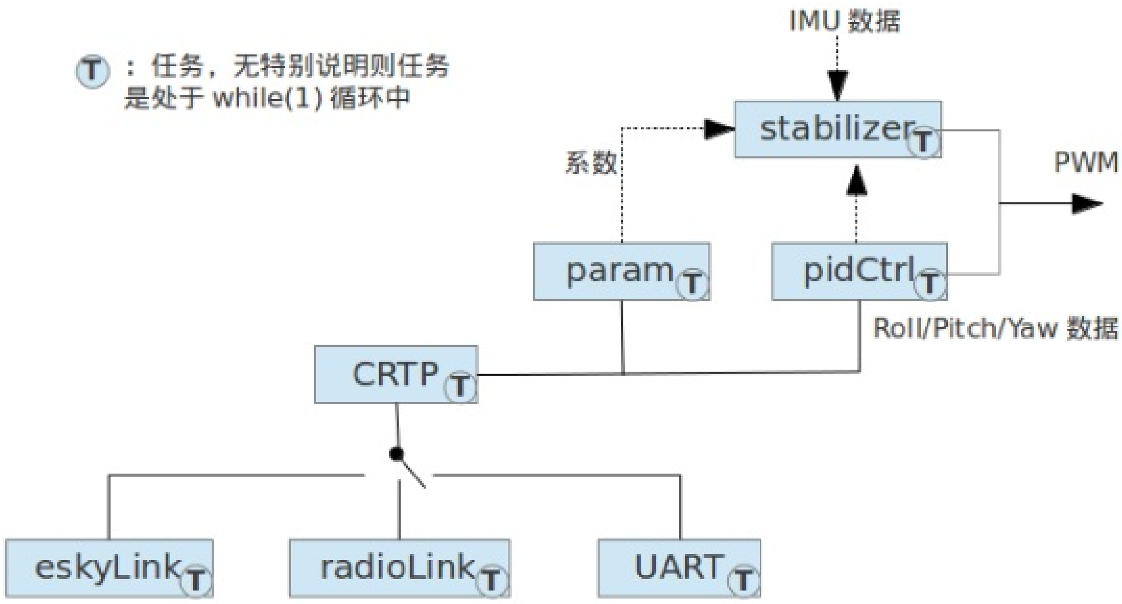
最关心的是系统中由多少个任务在运行。在FreeRTOS下，任务的初始化使用函数。所以，用grep命令参看一下源代码，可以看到一共有下面这些系统初始化的任务：

```
$ grep -inIw 'xTaskCreate' -r ./hal/ ./modules/
./hal/src/eskylink.c:312:  xTaskCreate(eskylinkTask, (const signed char * const)"EskyLink
./hal/src/pm.c:111:  xTaskCreate(pmTask, (const signed char * const)"PWRMGNT",
./hal/src/uart.c:145:  xTaskCreate(uartRxTask, (const signed char * const)"UART-Rx",
./hal/src/radiolink.c:237:  xTaskCreate(radiolinkTask, (const signed char * const)"RadioL
./modules/src/stabilizer.c:157:  xTaskCreate(stabilizerTask, (const signed char * const)"
./modules/src/crtp.c:77:  xTaskCreate(crtpTxTask, (const signed char * const)"CRTP-Tx",
./modules/src/crtp.c:79:  xTaskCreate(crtpRxTask, (const signed char * const)"CRTP-Rx",
./modules/src/info.c:68:  xTaskCreate(infoTask, (const signed char * const)"Info",
./modules/src/log.c:171:  xTaskCreate(logTask, (const signed char * const)"log",
./modules/src/pidctrl.c:43:  xTaskCreate(pidCrtlTask, (const signed char * const)"PIDCrtl
./modules/src/param.c:92:   xTaskCreate(paramTask, (const signed char * const)"PARAM",
./modules/src/system.c:68:  xTaskCreate(systemTask, (const signed char * const)"SYSTEM",
```

所有的任务在创建之后，都会进入`while(1)`的循环中，也就是任务一直循环运行。

任务之间的关系

那么任务之间的关系是怎么样的呢，如何各司其职完成系统的控制呢。下面使用一个简图进行了说明，逻辑主线是控制数据的传递。


T：任务，无特别说明则任务
是处于 while(1) 循环中
IMU 数据
stabilizer
系数
PWM
param
pidCtrl
Roll/Pitch/Yaw 数据
CRTP
eskyLink
radioLink
UART

通信部分介绍

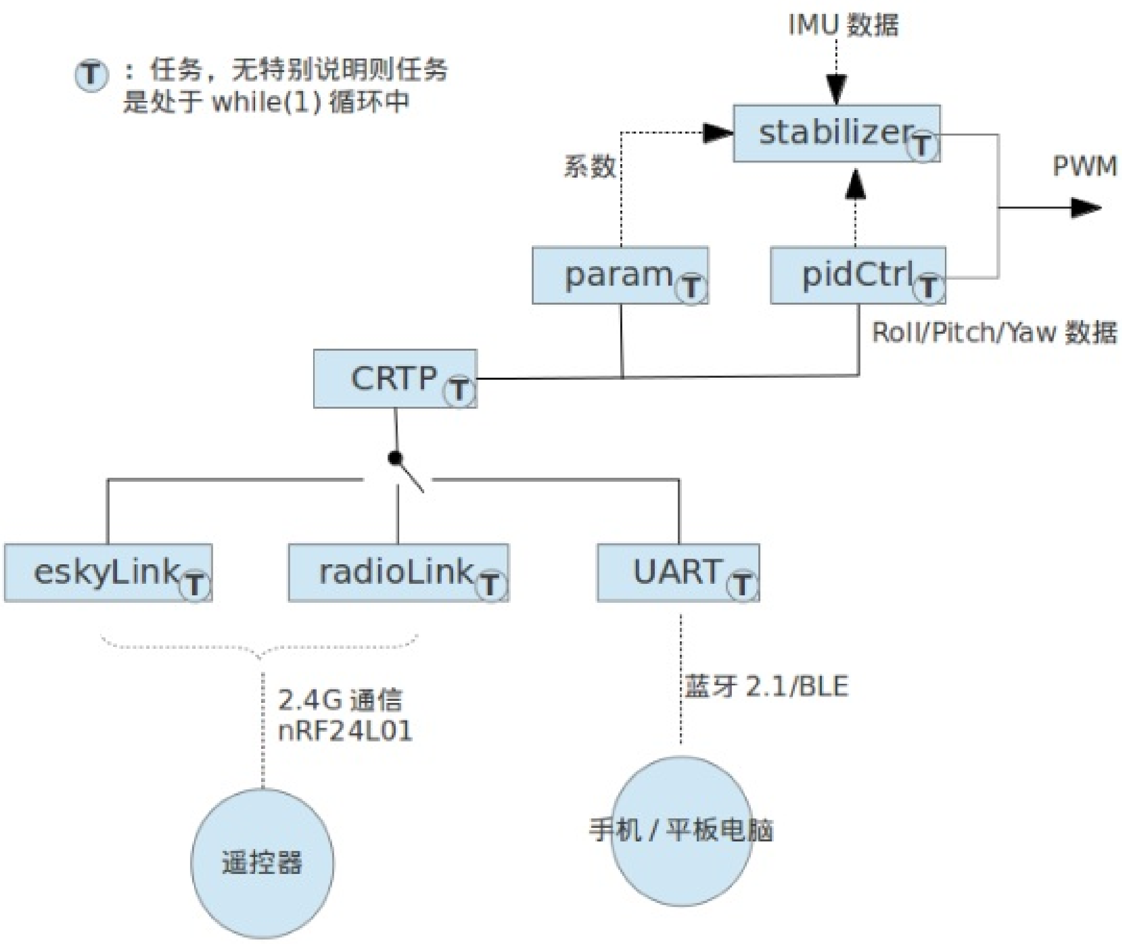
> 作者：nieyong

通信协议指的是遥控端和主控之间交互数据的封装，是一种自行约定的数据封装格式。我们采用的是Crazyflie项目中定义的CRTP协议。

遥控端和主控之间数据的交互，物理层可以有下面几种方式：

- 单片2.4G无线射频收发芯片，通过SPI接口和MCU连接
- 蓝牙2.1透传模块，通过串口UART和MCU连接
- 蓝牙4.0低功耗BLE透传模块，通过串口UART和MCU连接

这几种不同的物理层通信方式在同一时刻只能够选择其中的一种。并且蓝牙2.1和蓝牙BLE根据安装的透传模块不一样进行选择。



ESky Protocol

在上面示意图的的通信方式选择上，最左边为EskyLink，对应使用宏USE_ESKYLINK来开启是否选择该种链接方式。

```
# Make copter firmware to be used with the bootloader, CF controlled with eSky ET6i trans

$ make clean && make USE_ESKYLINK=1 CLOAD=1 all
```

所谓的Esky Protocol，其实就是ESky公司遥控器使用的通信协议。所以如果选用这种通信方式，那么可以使用ESky航模公司的遥控器进行控制。Crazyflie官网支持ESKY ET6I Remote Control遥控器。

关于ESky公司的2.4G遥控器设备的通信协议，参考ArduinoRCLib项目中的描述：

> The Esky 2.4 GHz equipment uses the Nordic NRF2401AG in both the transmitter and receiver. A compatible alternative to this chip is the NRF24L01+ (which is widely available).

ESky相关协议内容暂时不研究。

串口

使用串口作为物理层指的是蓝牙透传模块（包括蓝牙2.1或者蓝牙4.0 BLE模块）。串口配置为115200 8N1，收发异步。

一个可供主控解析的数据包格式，约定如下：

```
  7   6   5   4   3   2   1   0
  |     Port       |  Res. | Chan. |
  |             DATA 0            |
  :   :   :   :   :   :   :   :   :
  |             DATA 31           |
```

开始为一个字节的头，该字节中包括port字段，reserve字段，channel字段。紧跟着该自己，就是0-31个字节的数据，数据的长度是任意的。

头字节中的port字段用来区分该数据包的功能，常见的包括下面几个

- 0x0 : console
- 0x2 : parameter，表示后面的数据是系统的参数
- 0x3 : commander，操作命令，控制飞机的飞行，包括pitch/yaw/roll等
- ……

对于串口传输，在上面数据包的最前面需要加上两个字节0xaaaa的起始数据，在最后加上一个字节的checksum数据。格式如下：

```
  7   6   5   4   3   2   1   0
  |             0xAA            |
  |             0xAA            |
  |      Port      |  Res  | Chan. |
  |         Packet length        |
  |            DATA 0            |
  :   :   :   :   :   :   :   :   :
  |            DATA 30           |
  |            Cksum             |

  |  0xAA   |   0xAA   | Header | Length | Data0  | Packet | Cksum  |
```

注意，紧跟在头后面的第一个data字节，表示后面数据的长度。

操作命令

操作命令是指用于控制飞行器起飞，前后左右运动的命令，英文commander。操控命令是遥控器最基本，也是最常用的命令。数据包头中port字段为0x3表示操作命令。Crazyflie操作数据格式约定如下。

```
| ROLL    | PITCH |   YAW   |THRUST |
0       4       8       12      14 bytes
```

操作数据一共14个字节，前12个字节分别表示Roll，Pitch，Yaw的值，每个值使用4个字节。后2个字节表示Thrust的值，使用2个字节。

所以，使用串口发送的一个操作命令示例如下：

```
0xaa 0xaa 0x30 0x0e 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

其他

这里记录了Crazepony开发过程中的部分内容，例如板形轮廓，文件编码等等。

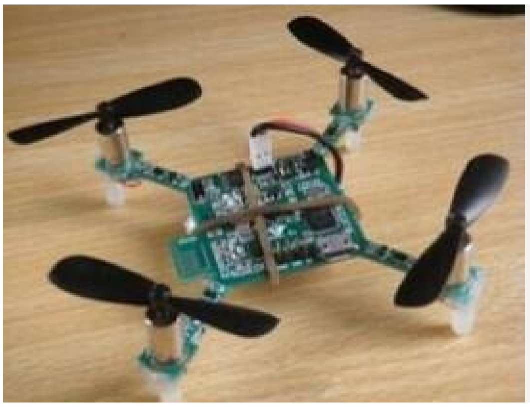
Crazepony的板形轮廓及沉金工艺

> 作者：陈泽利，nieyong

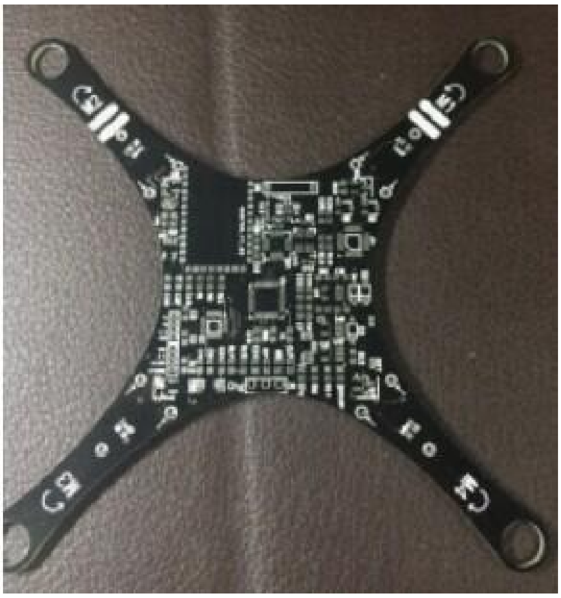
板形轮廓

目前，市面上的绝大部分的小四轴采用的是直“X”形和直“十”字的结构。但这两种结构有一个缺点，就是机臂的问题。如果机臂做得太宽，毫无疑问会严重影响到整个小四轴的美观性。但是如果不做宽，为了给螺旋桨让出足够的空间，机臂是要伸出小四轴主体一段相对比较长的距离的，这样的话就会造成整个机臂细长，有碰撞或者受力稍大都可能造成机臂断掉（在网上看到不少四轴爱好者就抱怨过这个问题）。





对以上的问题，Crazepony在设计过程中，淘汰了直“X”和直“十”字的结构。采用了全新的双曲线轮廓（注意：机臂与机臂之间并不是半圆），从小四轴主体到机臂末端沿一条曲线变小，这样既保证了机臂的牢固性，又保证了整体的美观。采用双曲线结构之后，整机的流畅性显得淋漓尽致，给人一种柔和圆滑的感觉，与直“十”字的那种呆板简直天差地别。



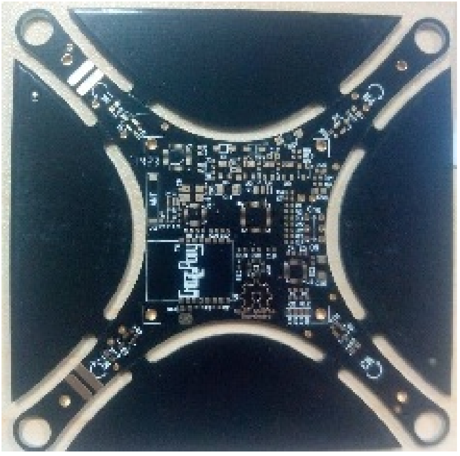
Crazepony的双曲线结构同时，有别于传统小四轴飞行器PCB板结构的一个地方在于，Crazepony在机臂末端做了一个和电机外径同等大小的圆孔，将电机安装在圆孔内，就相当于将电机栓在了小四轴这个整体上面。这样的方式与市面上绝大多数的小四轴将电机安装在PCB板外的方式相比有一个很明显的好处，就是：避免了电机支架松动，导致电机晃动，从而引起在飞行过程中整体的晃动。

PCB板工艺

Crazepony采用的是黑色PCB板加沉金工艺。选择黑色PCB板的主要原因就是觉得好看！搭配上银色的铅锡焊盘之后又显得有点违和，出于对整体的美观及进一步的工艺要求，Crazepony在第四版的时候，将原来普通的银色铅锡焊盘改为了沉金工艺的金色焊盘。沉金工艺的好处在于以下几点：

- 沉金的金黄色配上PCB板的黑色，有木有一种高大上的感觉！！！
- 在焊接时，沉金工艺相对于普通的铅锡焊盘更容易焊接，不易造成焊接不良。
- 沉金晶体结构致密，不易产生氧化。同时表面的平整度与待用寿命较好。
- 沉金板焊盘上只有镍金，线路上的阻焊与铜层的结合更牢固。

想对沉金工艺进一步了解的朋友可以参考PCB沉金工艺介绍。



源代码的文件编码和文件格式

> 作者：nieyong

文件编码

原则上所有源代码默认使用utf-8编码。

网站源码文件必须使用utf-8编码。如果使用其它的编码，例如中文中常见的gdb格式，会导致中文乱码，或者网页无法生成的问题。

文件编码常识

Windows操作系统默认文件编码是gbk，又叫做gb2312或者cp936。cp936是微软发布的用在文件系统中的编码方式，而gb2313是中国国家标准，又叫做ANSI格式编码。例如在notepad++编辑器上gb2312编码的文件就显示为ANSI格式编码；使用Linux下的file命令，则显示为ISO-8859；在Vim下使用 `set fenc?` 命令查看，则显示为fileencoding=euc-cn。其实这三种都是表示gb2313编码。

Linux操作系统默认文件编码是utf-8，一般用系统宏$LANG表示。

```
$ echo $LANG
zh_CN.UTF-8
```

下面几个crazepony固件代码文件，就是使用的Windows下的默认编码gb2313。为了在某些工具下（例如gitk工具）中文不显示为乱码，我们要求所有的文件都使用utf-8编码。但是由于我们现在使用的Keil 4不支持utf-8编码，所以utf-8编码的中文在Keil 4下面会显示为乱码，所以这部分固件代码没有使用utf-8编码。

```
$ file User_Src/*
User_Src/main.c:           ISO-8859 C++ program text
User_Src/Sys_Fun.c:        ISO-8859 C program text
User_Src/Sys_Fun.h:        ISO-8859 C program text
……
```

行结束符问题

在Linux/Windows/Mac下，行结束符不一样。这样在多个平台之间进行协作开发时，在checkin和checkout之后，会有很多行结束符不统一引起的问题。

添加.gitattribute文件，可以解决行结束符在多个平台下不一致的问题。

```
# Explicitly declare text files we want to always be normalized and converted
# to native line endings on checkout.
* text

# Denote all files that are truly binary and should not be modified.
*.png binary
*.jpg binary
*.gif binary
*.jar binary
*.so  binary
style/fonts/* binary
style/images/* binary
```

上面为现在使用的.gitattribute配置，解释如下：

- 代码库中都为LF，也就是checkin的时候，会把所有文件的行结束符转化为LF；
- 工作路径下为所在OS的行结束符，也就是在checkout的时候，git回自动根据当前的OS将文件的行结束符做转化；
- 对于图片（以png/jpg/gif等结尾），jar库，和so文件等二进制文件，不进行转化；

关于本书

> 作者：nieyong

本书《Crazepony开源四轴飞行器》使用了GitBook管理和发布。是开源四轴飞行器项目Crazepony配套书籍。

Gitbook

Gitbook是一个基于Node.js的命令行工具，可使用Github/Git和Markdown来制作精美的电子书。

Web版本书籍调试命令：

> gitbook serve ./

在浏览器下输入http://0.0.0.0:4000就可以查看修改效果。

Web版本书籍生成命令：

> gitbook build ./ --output=./outputFolder

制作说明

《Crazepony开源四轴飞行器》电子书的内容来自Crazepony网站百科wiki，这里只进行了整理和发布，给大家提供在线查看的Web版本以及可供下载的PDF版本。

Web版本书籍调试命令：

> gitbook serve ./

在浏览器下输入http://0.0.0.0:4000就可以查看修改效果。

Web版本书籍生成命令：

> gitbook build ./ --output=./outputFolder

在Crazepony网站的Book目录下的就是使用该命令生成的静态Web版本书籍。

书籍内容来自Crazepony网站wiki，我们会定期将wiki的更新同步到书籍中来。虽然网站wiki和gitbook书籍都是使用markdown语言写成，但是还是有部分头文件不相同。我们专门写了一个python脚本批量将网站wiki的md文件转化为适合gitbook的文件。进入./wiki目录，运行：

> ./deleteline.py

版权说明

《Crazepony开源四轴飞行器》全部内容以Creative Commons署名-非商业性使用-相同方式共享3.0协议发布。任何个人或者组织的商业用途都是违法的。

知识共享是一个非营利组织，该组织提供同名的一系列著作权许可方式。知识共享组织的主要宗旨是使得著作物能更广为流通与改作，作为其他人据以创作及共享的基础，并以所提供的许可方式确保上述理念。

联系我们

邮箱： `info@crazepony.com`

发布版本记录

最新版下载地址（PDF）：

- 2015-2-7， 第一次发布，0.1版本。
- 2015-11-24， 第二次发布，0.2版本。